3122

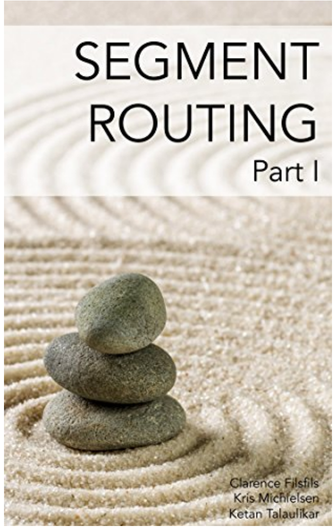
Segment Routing: Technology deep-dive and advanced use cases

Clarence Filsfils – Cisco Fellow

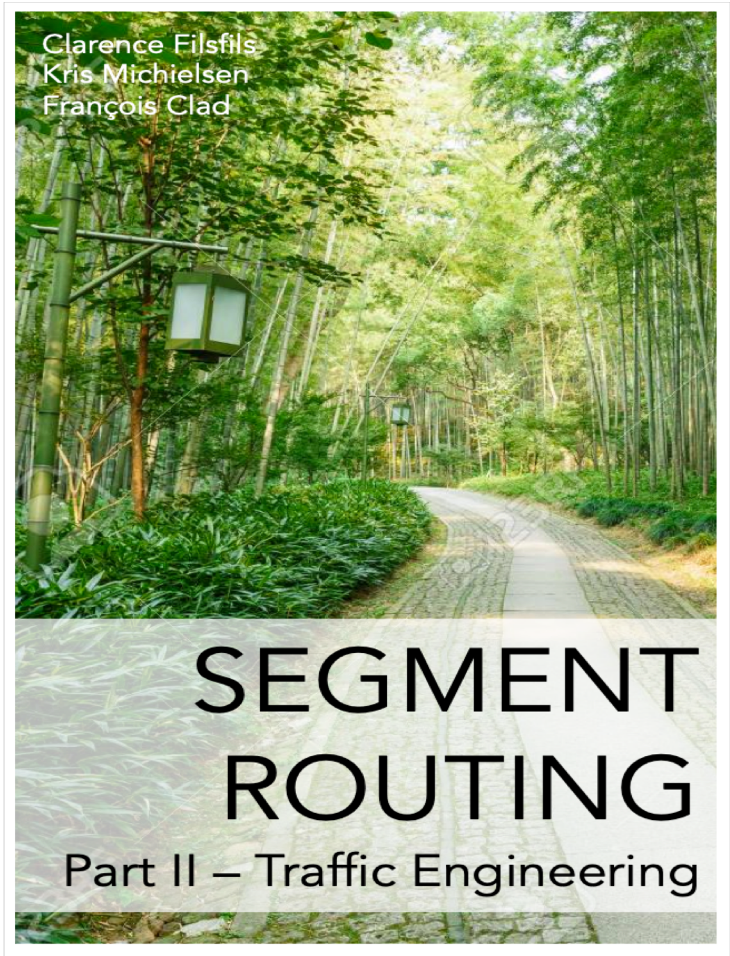
Acknowledgements

- François Clad
- Kris Michielsen
- Jose Liste
- Alberto Donzelli
- Jakub Horn
- All the SR team

Part II is available!



amzn.com/B01I58LSUO



amazon.com/dp/B07N13RDM9

Agenda

- 40 min: SR-MPLS: new solutions
- 40 min: SRv6: new solutions

Highlight the key new concepts

We could spend 4 hours on this
Details are in the slides

- 20 min: Anton Karneliuk, Vodafone
  - 1st ever deployment: Min Latency Slice and Bounded Latency
- 20 min: Jose Liste
  - Demo: SDWAN with SR-SLA underlay differentiation

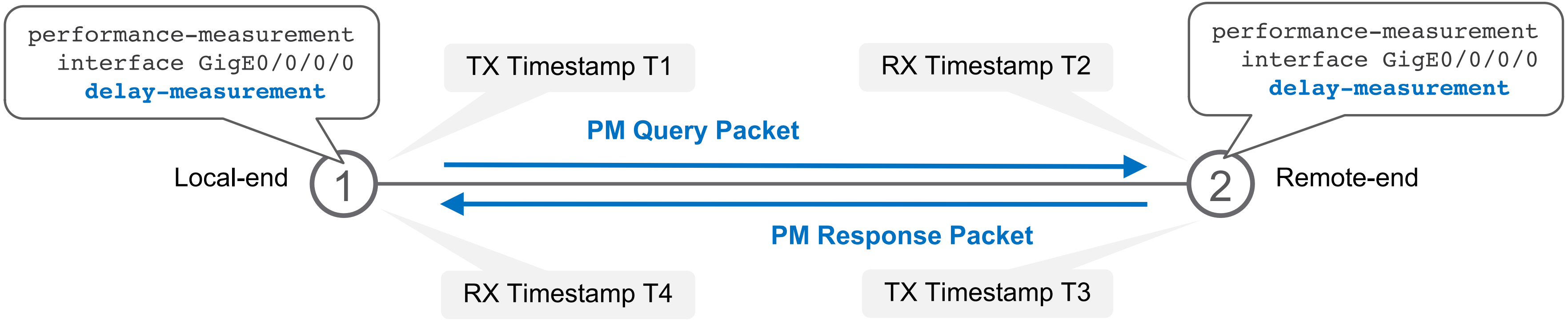
Illustration

SR MPLS

Per-Link Delay Measurement

Cisco live!

Link Delay – Probe Measurement



- One Way Delay = (T2 – T1)
- Two-Way Delay = ((T2 – T1) + (T4 – T3)) / 2
- Timestamps added in hardware
- PM Query message using RFC 6374 packet format with MPLS/GAL or IP/UDP Encap

Default: every 3 sec

Per measurement interval

- Probe every 3sec
- Over a measurement interval (default 30sec)
  - minimum
  - average
  - maximum
  - variance

SRTE handles Minimum delay (propagation delay)

- Minimum delay provides the propagation delay
  - fiber length / speed of light
- A property of the topology
  - with awareness of DWDM circuit change
- SRTE (Policy or Flex-Algo) can optimize on min delay

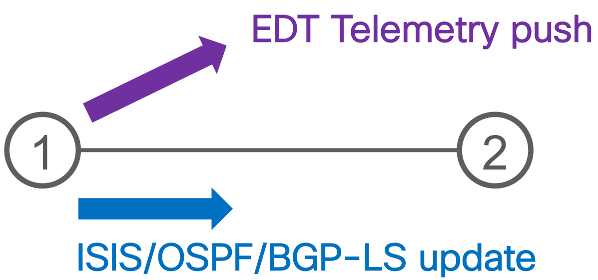
QoS handles Average, Max and Variance (buffer)

- Depends on buffer congestion
  - (traffic burst over line rate) / line rate
- Highly variable at any time scale
- Not controlled by routing optimization
- Controller by QoS
  - Priority queue, WRR, WFQ…
  - Tail-Drop, RED…

Routing stability – Telemetry accuracy

Every 30sec
(10 queries)

Every 120sec
IF significant min change
THEN trigger an ISIS/OSPF flood



ISIS, OSPF, and BGP-LS signaling

- Advertise extended TE metrics – e.g. link delay (in usec)
  - Unidirectional Link Delay
  - Minimum and Maximum Unidirectional Link Delay
  - Unidirectional Link Delay Variation

- RFC 7810 (IS-IS)
- RFC 7471 (OSPF)
- draft-ietf-idr-te-pm-bgp (BGP-LS)

Leveraged by SRTE

- SR Policy for min delay

```
segment-routing
  traffic-eng
    policy FOO
      color 20 end-point ipv4 1.1.1.3
      candidate-paths
        preference 100
          dynamic
            metric
              type delay
```

- IGP SR Flex-Algo for min delay

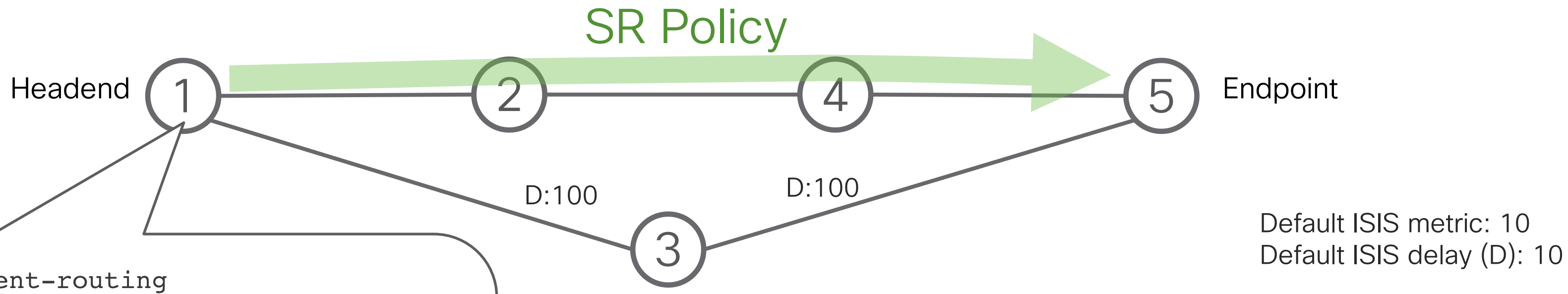
```
router isis 1
  flex-algo 128
    metric-type delay
```

Shipping and in deployment ☺

More details from Anton / VF

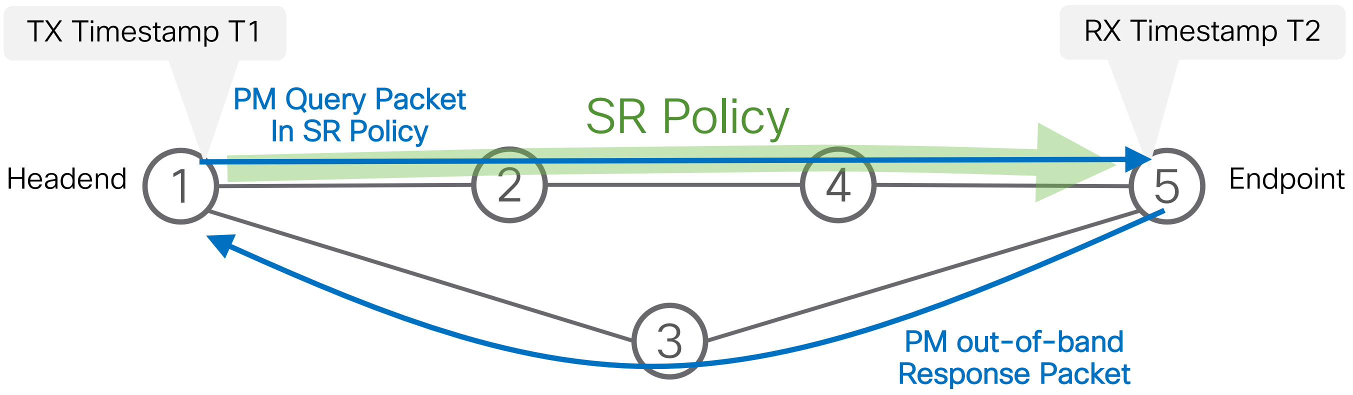
Per-Policy Delay Measurement

Configuration



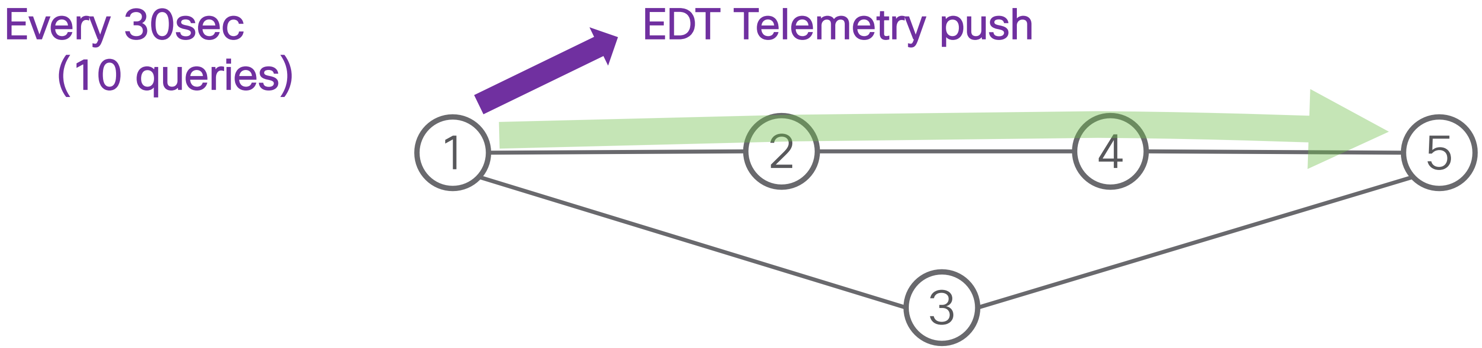

```
segment-routing
 traffic-eng
  policy FOO
   color 20 end-point ipv4 1.1.1.5
   delay-measurement
   candidate-paths
    preference 100
     dynamic
      metric
       type delay
```

Probe Measurement



- One Way Delay = (T2 – T1)
  - Requires clock synchronization
- Default: Send Query every 3 sec
- PM packets same format as link delay measurement

Telemetry



- Use telemetry to collect the evolution of the end-to-end delay metrics at fine time scale (min, max, avg at probe period)
- In first release, no routing reaction to excess measured latency

Ask a demo at SR booth

SR IGP Flex-Algo

Flexible Algorithm

- We call “Flex-Algo”
  - An algorithm defined by the operator, on a per-deployment basis
- Flex-Algo K is defined as
  - The minimization of a specified metric: IGP, TE or delay
  - The exclusion of certain link properties: affinity or SRLG
- Example
  - Operator1 defines Flex-Algo 128 as “minimize IGP metric and avoid link-affinity “green”
  - Operator2 defines Flex-Algo 128 as “minimize delay metric and avoid link-affinity “blue”

Flex-Algo Participation and Prefix-SID

- Each node MUST advertise the Flex-Algo(s) that it participates in
  - A Flex-Algo K can be enabled on all or a subset of nodes
  - Each node can participate in multiple Flex-Algos

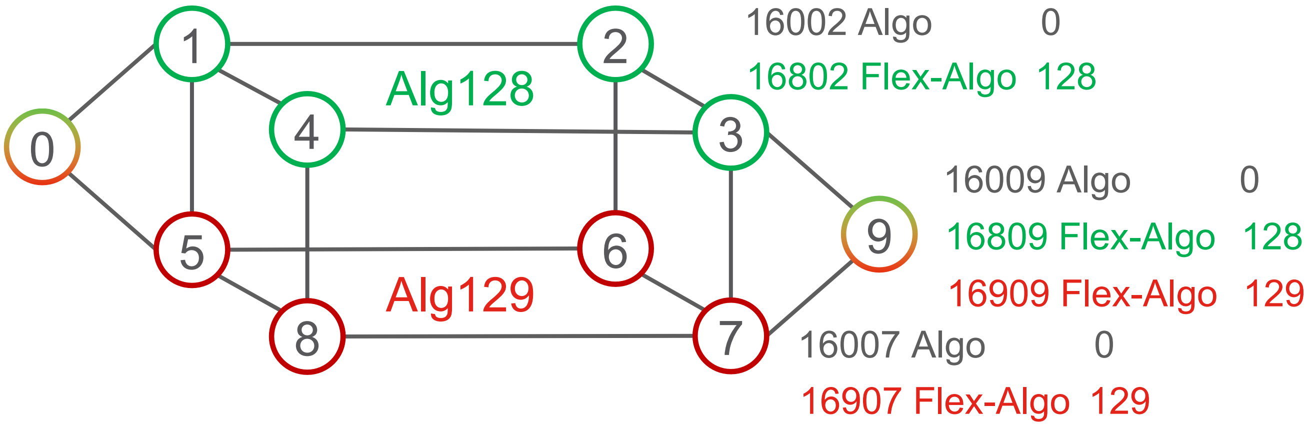
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type       |     Length    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Algorithm 1   |  Algorithm 2  | Algorithm ... |  Algorithm n  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Likely it also advertises prefix SIDs for these Flex-Algos

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type       |    Length     |     Flags     |   Algorithm   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  SID/Index/Label (variable)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

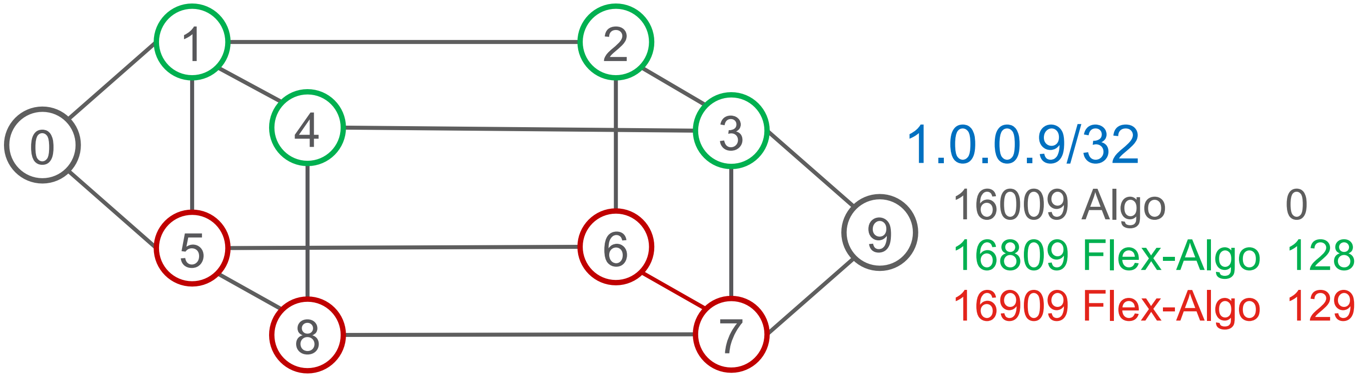
Example: 3 planes: 0 / 128 / 129

- All nodes support standard Algo 0
- Green nodes advertise support for Flex-Algo 128 as "Minimize IGP metric"
- Red nodes advertise support for Flex-Algo 129 as "Minimize IGP metric"
- Each node k advertises a Prefix SID for every Algo it supports:
  - 1600k for Algo 0
  - 1680k for Algo 128
  - 1690k for Algo 129



No additional loopback address

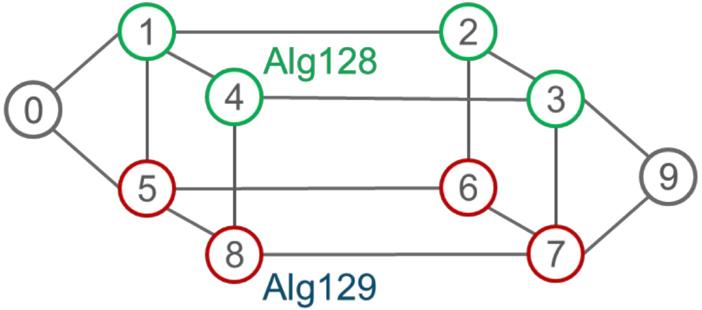
- Flex-Algo Prefix SIDs can be advertised as additional prefix-SIDs of the existing loopback address



Flex-Algo Definition

- Each node MUST have the definition of the Flex-Algo(s) that it is participating in
  - e.g. Flex-Algo 128: minimize on IGP metric and avoid TE affinity RED
- Local configuration
  - likely automated via a solution such as NSO
- Learned from a central entity via ISIS flooding
  - new top TLV defined for Flex-Algo definition advertisement

Algo 128: minimize IGP metric
Algo 129: minimize IGP metric



Computation

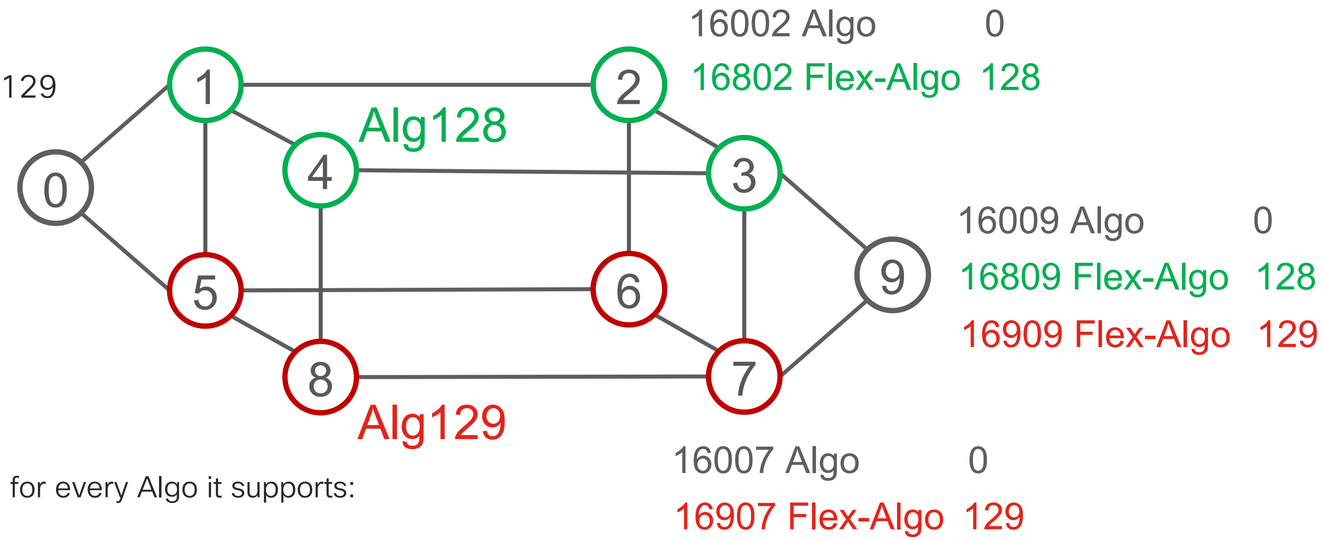
- A node N computes Flex-Algo K if
  - it is enabled for K, and
  - it has a consistent definition for K
- If so, the first step is to define the topology of K
  - N prunes any node that is not advertising participation to K
  - N prunes any link that is excluded by the algorithm of K
    - e.g. if K excludes TE-affinity RED then any link with TE-affinity RED is pruned
  - The resulting topology is called Topo(K)
- The second step is to compute shortest-path tree on Topo(K) with the metric defined by K
  - it could be the IGP metric, the TE metric or the delay

FIB installation

- The third step is to install any reachable Prefix-SID of Flex-Algo K in the forwarding table along the computed shortest-path on Topo(K)

Summing up the config

- Grey nodes support Algo 0/128/129
- Green nodes support 0/128
- Red nodes support 0/129
- Algo 128: minimize IGP metric
- Algo 129: minimize IGP metric



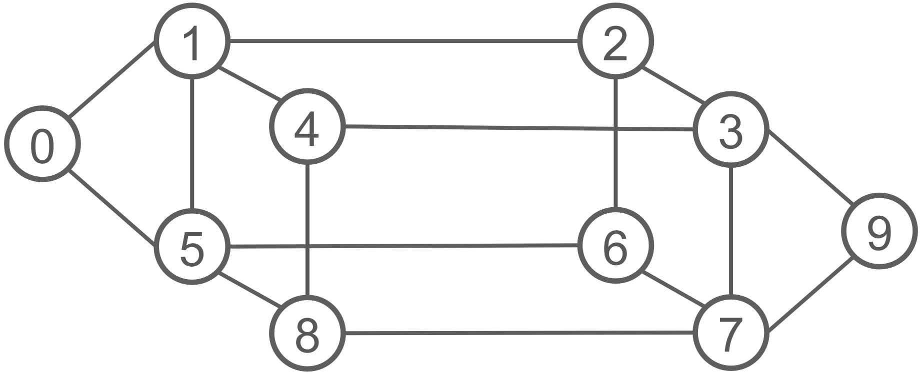
- Each node K advertises a Prefix SID for every Algo it supports:

  E.g. 1600K for Algo 0

  1680K for Algo 128

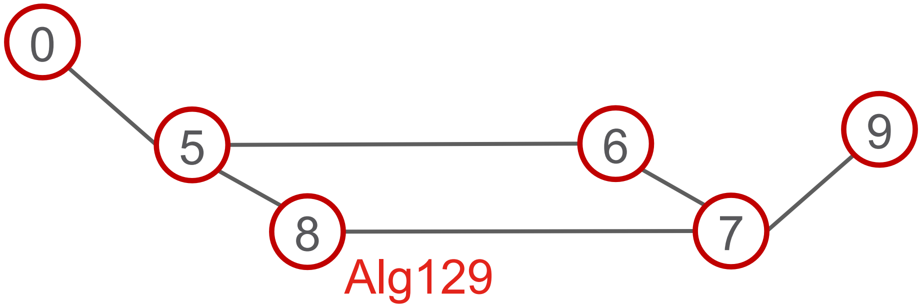
  1690K for Algo 129

Topo(0)

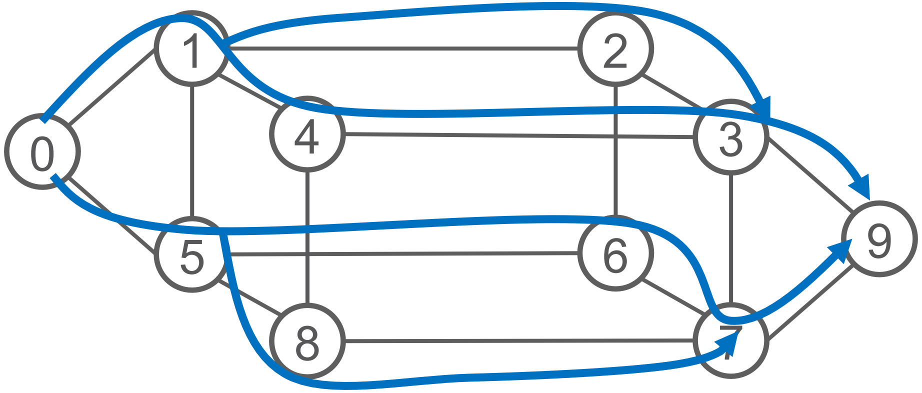


Topo(128)

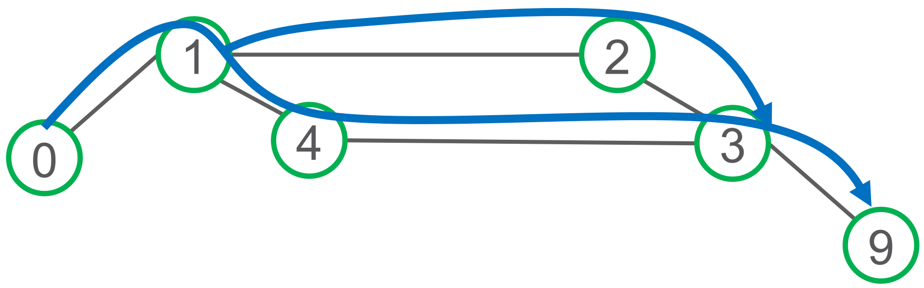
Topo(129)



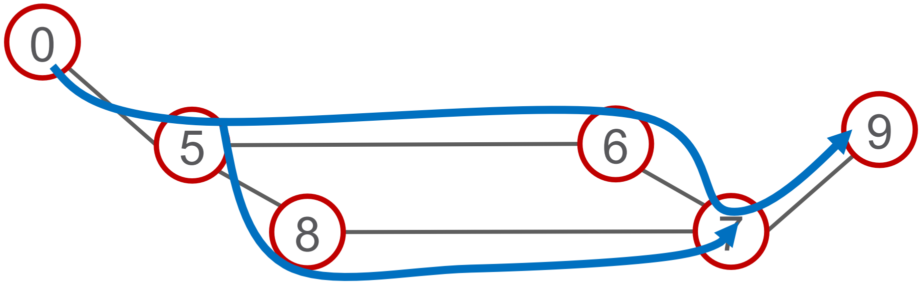
Prefix-SID 16009 of Algo 0



Prefix-SID 16809 of Flex-Algo 128



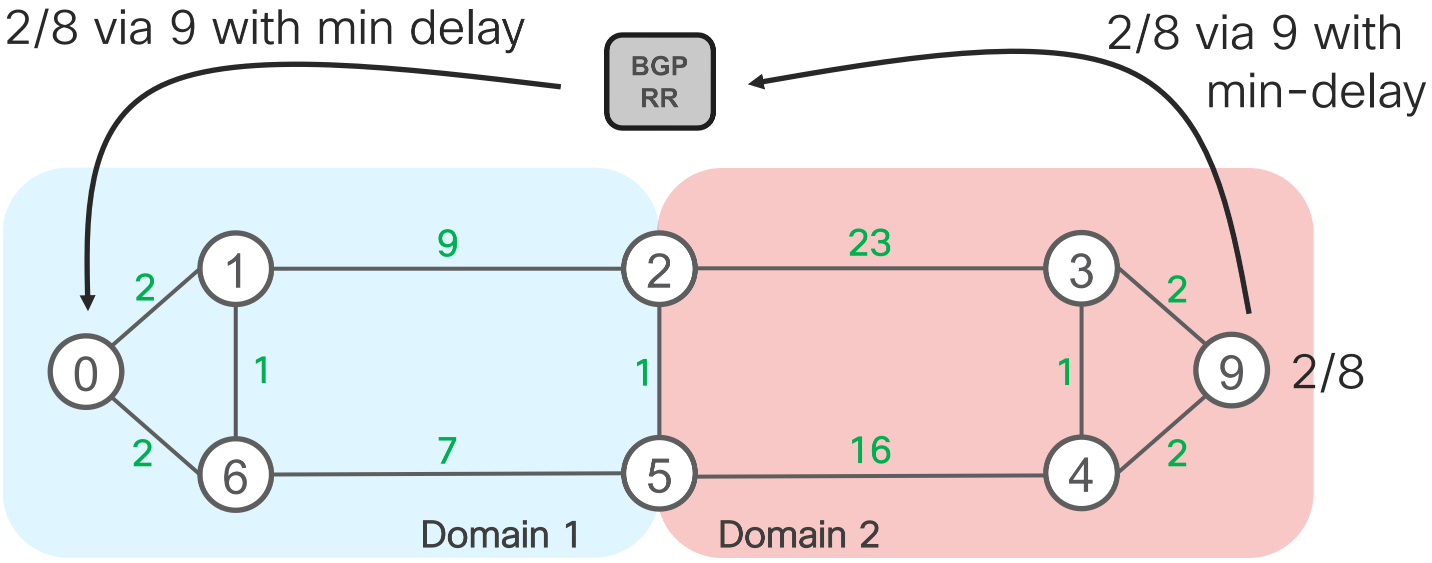
Prefix-SID 16909 of Flex-Algo 129



TI-LFA

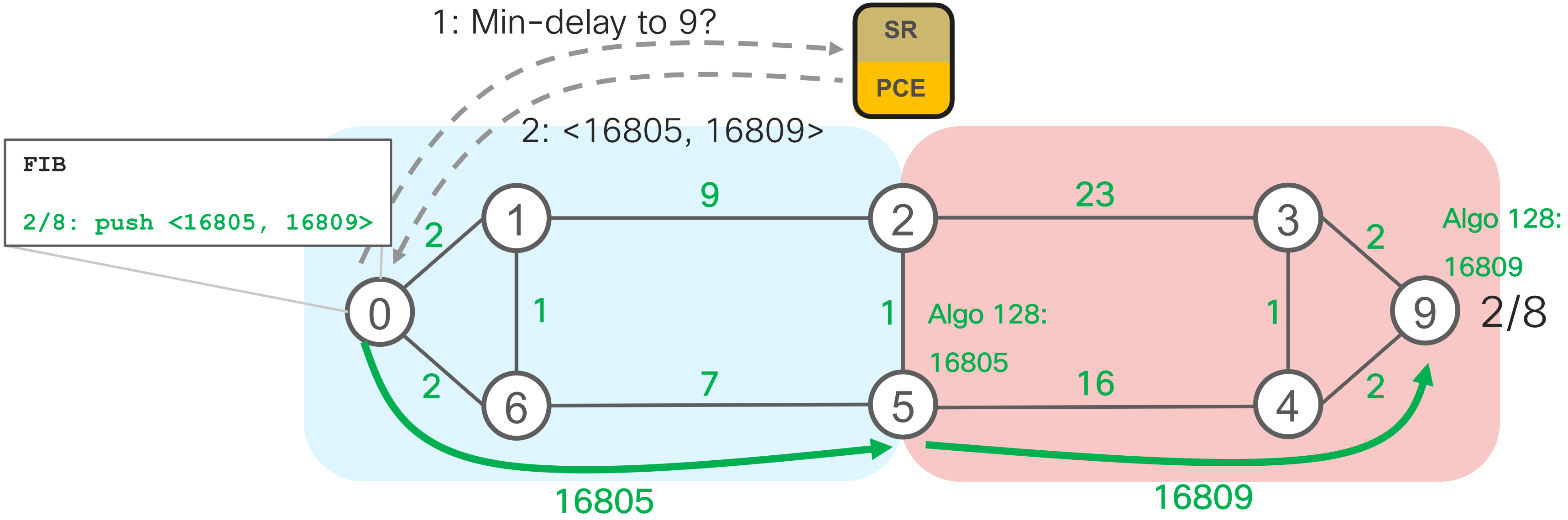
- The TI-LFA algorithm is performed within Topo(K)
- The backup path is expressed with Prefix-SIDs of Algo K
- Benefits: the backup path is optimized per Flex-Algo!

ODN and AS Inter-Domain delay



- Any node of both domain supports Algo 128
- Algo 128 is defined as min-delay
- The delay of each link is reported in drawing
- The IGP metric per link is 10
- 9 advertises 2/8 with color 100

ODN and AS Inter-Domain delay – Cont.



- Upon receiving 2/8, node 0 dynamically creates an SR Policy to 9
- As 9 is beyond its domain, node 0 requests the computation from its PCE
- PCE replies with MPLS stack <16805, 16809>, leveraging the Flex-Algo(128) SIDs

Shipping and in deployment ☺

More details from Anton / VF

Cumulative-Metric Bound

Cisco live!

3-tiered latency service

- Minimum Latency
  - ISIS Flex-Algo 128 ⇔ minimize the delay to the endpoint
- Minimum Cost
  - ISIS Algo 0 ⇔ minimize the isis metric to the endpoint

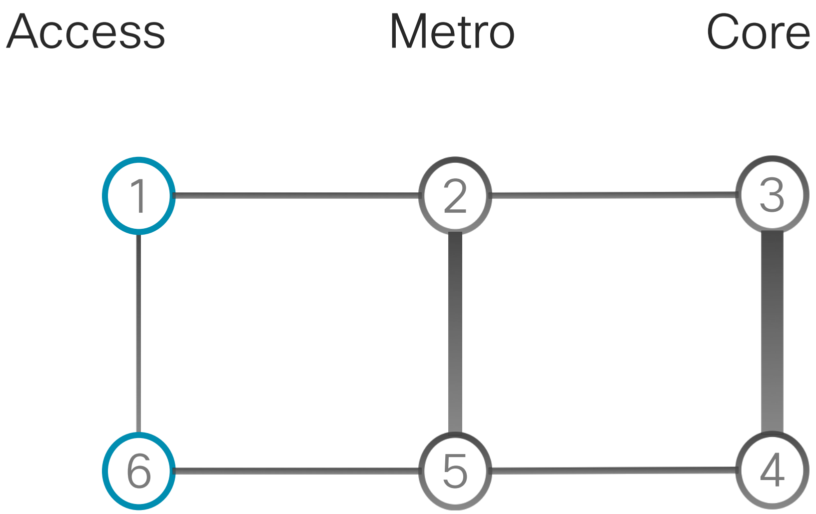
3-tiered latency service

- Minimum Latency
  - ISIS Flex-Algo 128 ⇔ minimize the delay to the endpoint
- Minimum Cost
  - ISIS Algo 0 ⇔ minimize the isis metric to the endpoint
- Minimum Cost but with a bound on the delay
  - SRTE Policy
  - SR Native Algorithm
  - Minimize IGP metric
  - Cumulative delay along the path <= delay bound

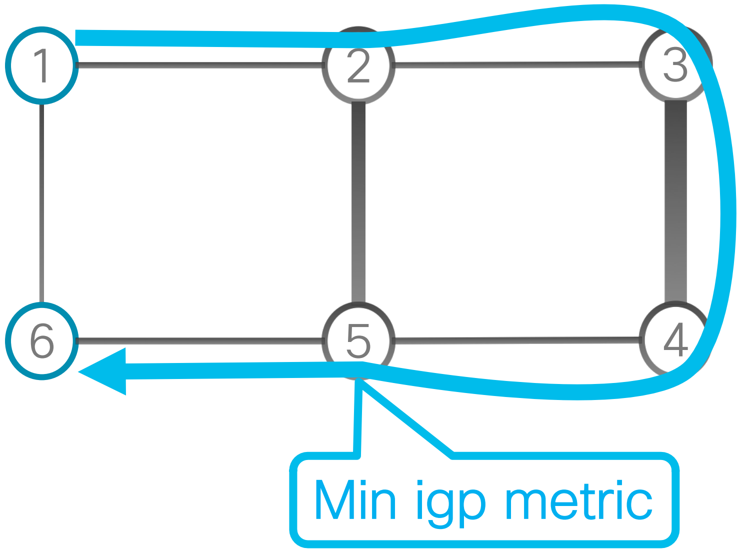
Another obvious business service that was never realized before SR

Business Relevance: Cost vs Latency

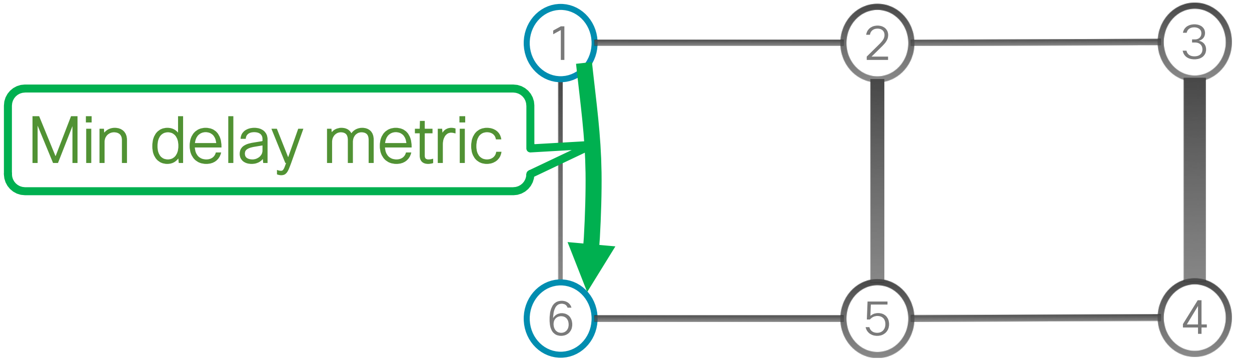
- Core
  - Cheaper
  - Longer delay
- Access Shortcut
  - Most expensive
  - Lowest latency
- Metro Shortcut
  - In-between



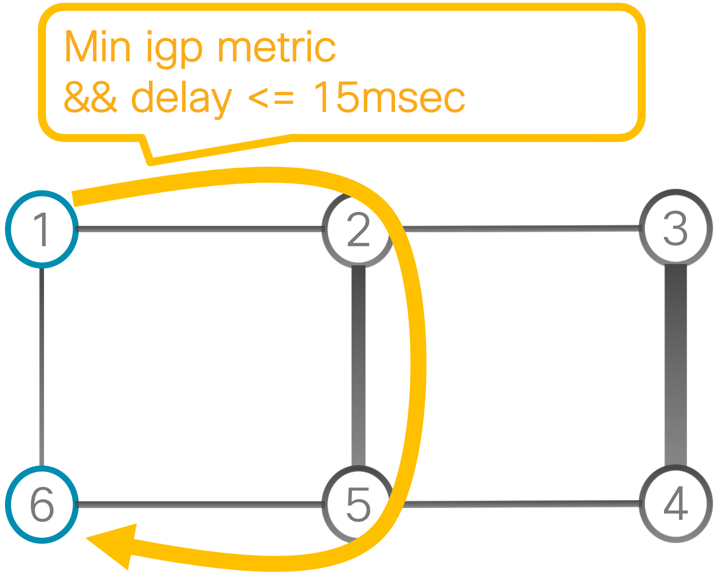
Bulk of the traffic - Best-Effort



Ultra Low Latency – 5G



Business traffic with constraint



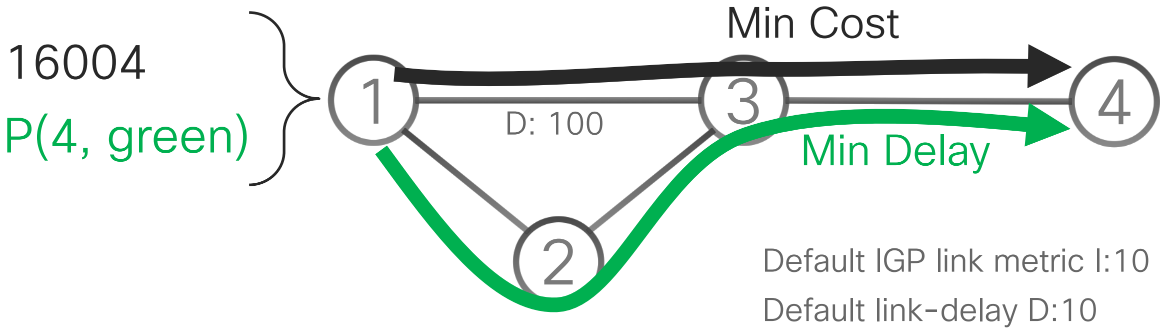
Shipping and in deployment ☺

More details from Anton / VF

Per-Destination ODN/AS

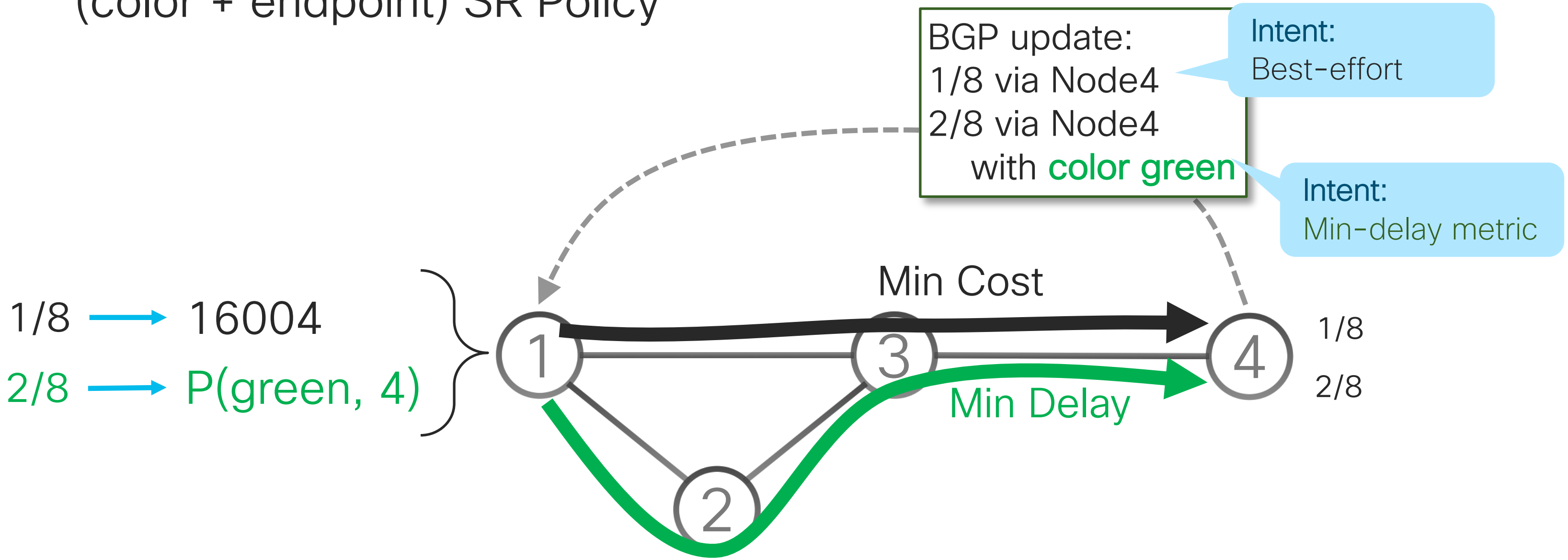
Setting the example

- ISIS Prefix-Segment to 4
  - 16004 along the min-cost path
- @1: SR-TE (4, green)
  - Defined as minimization of the delay
  - Result of the native SR algorithm: <16002, 16004>



Per-destination Automated Steering

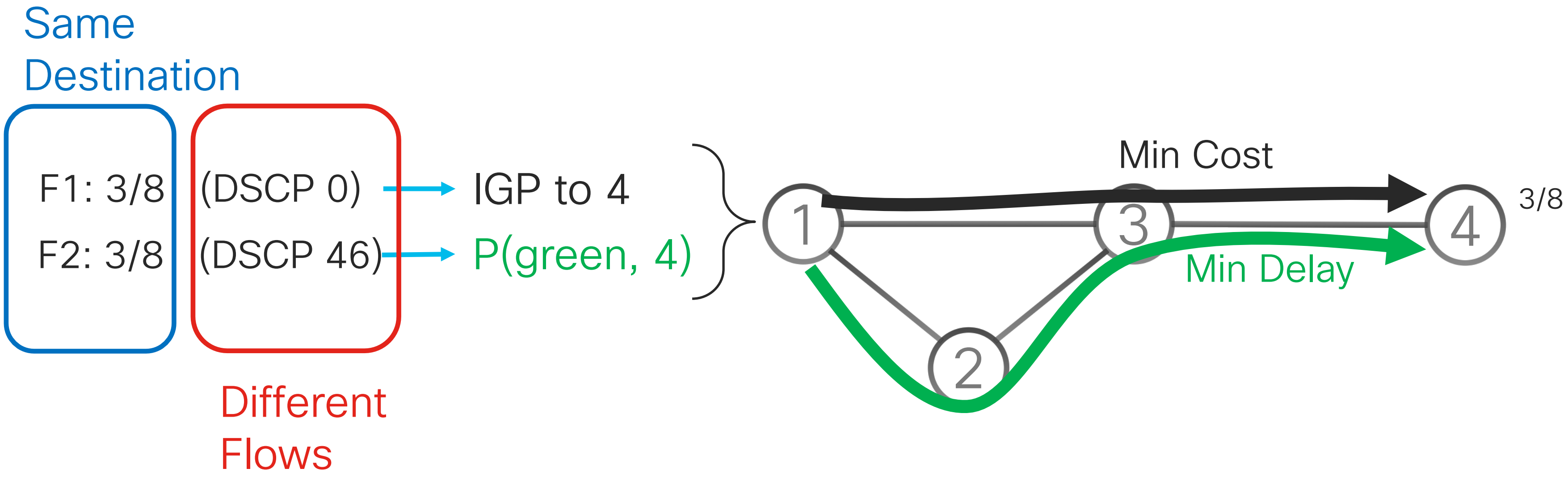
- Automated Steering steers service routes on their matching (color + endpoint) SR Policy



Per-Destination ODN

- Our solution is even richer
- When the colored BGP route is received, the SRTE policy is dynamically instantiated on-demand
- The color is associated with an SRTE SLA objective

Need for Per-Flow ODN/AS



Per-Flow ODN/AS

Forward-Class

- FC: a local value attached to a packet **within a router**
  - Range from 0 to 7
- **Set on the ingress interface on the basis of 5-tuple ACL or DSCP**

```
class-map type traffic match-any MinDelay
 match dscp 46
end-class-map
!
class-map type traffic match-any PremiumHosts
 match access-group ipv4 PrioHosts
end-class-map
!
```

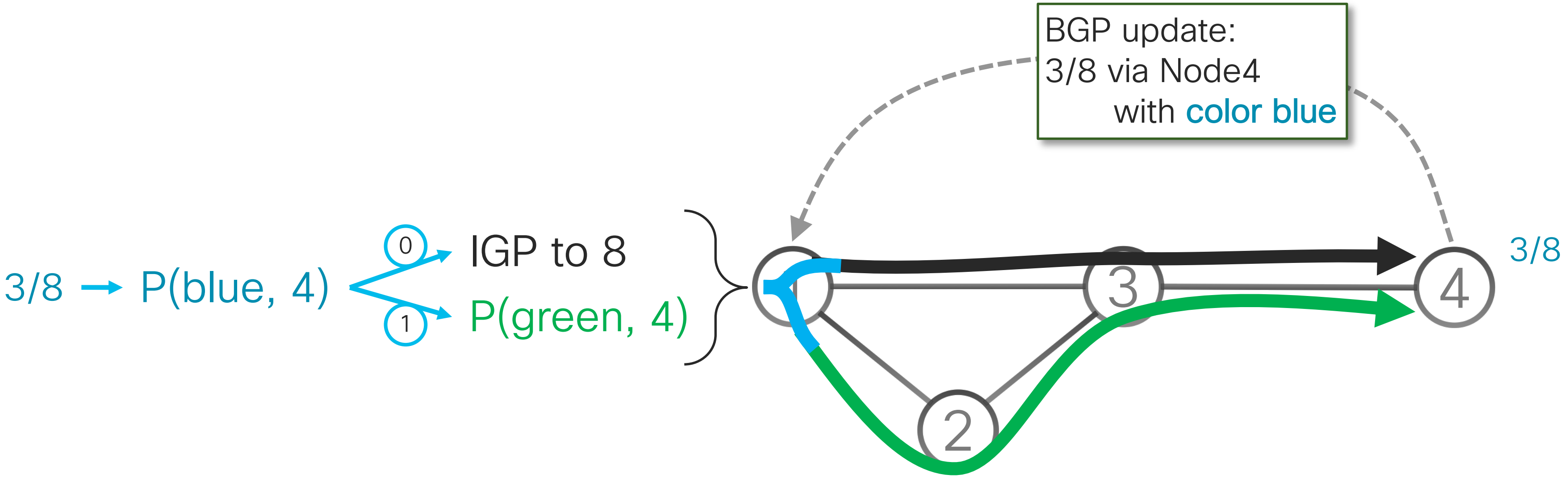
```
policy-map type pbr MyPerFlowPolicy
 class type traffic MinDelay
  set forward-class 1
 !
 class type traffic PremiumHosts
  set forward-class 2
 !
 class type traffic class-default
  set forward-class 0
 !
end-policy-map
```

Per-Flow SR Policy

- Identified as (endpoint, color)
- Single Color space shared by all policy
  - per-destination: e.g. [100-199]
  - per-flow e.g. [200-299]
- A Per-Flow SR Policy provides up to 8 “ways” to the endpoint
- The FC setting of the packet selects the “way”
- The “way” can be a per-Destination Policy or a classic RIB path (IGP)

Per-Flow Automated Steering (AS)

- AS automatically steers a service route on the policy (E, C):
  - E == next-hop
  - C == color of the service route

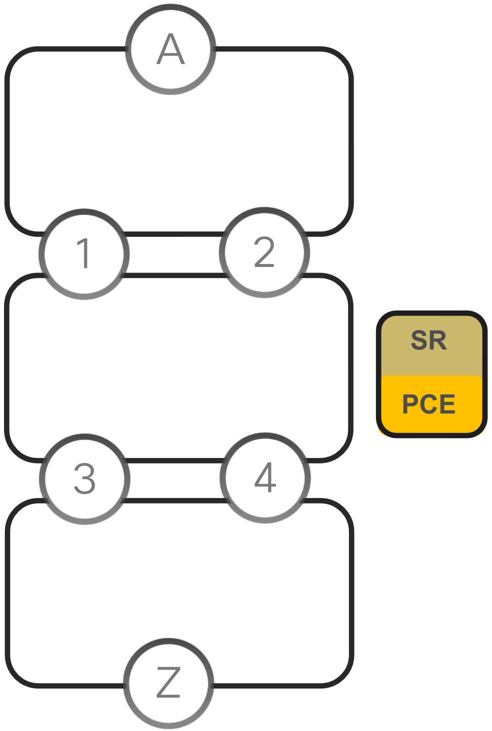


More details during SDWAN demo

Using Anycast-SIDs

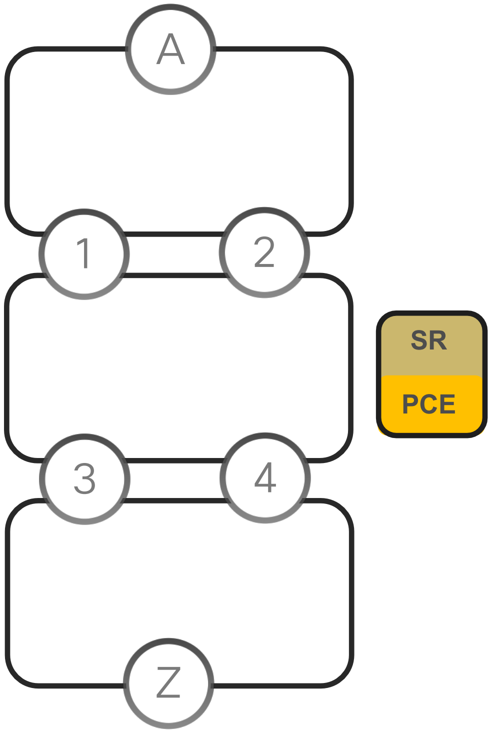
Example – 3 Isolated domains

- No IGP redistribution
- No BGP3107
- 1 and 2 share anycast 16012
- 3 and 4 share anycast 16034
- A receives a VPN route with nhop Z
- A resolves the SR path to Z via
  - ODN/AS
  - SR PCE



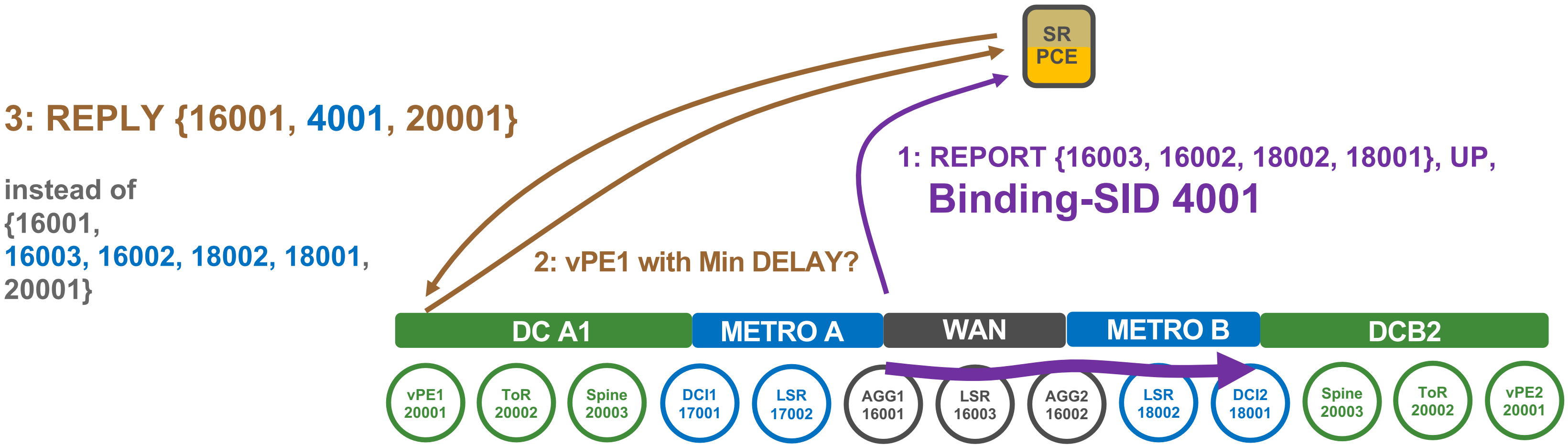
Obvious requirement

- SR PCE’s native SR algorithm uses anycast SID as much as possible
- If the distances to/from 1/2 and 3/4 are equal
- <16012, 16034> is better than
  - <16001, 16003> and any other variation
- Why?
  - More load-balancing
  - More resiliency



Hierarchical ODN (H-ODN)

Leverage Transit Policies on end-to-end path



- SR PCE leverages existing SR policies for transit purposes by including their Binding-SID as part of a SID-list for an end-to-end SR Policy
  - This creates a hierarchy of SR Policies
- Shorter SID list and churn isolation between domains
  - Even if the WAN-MetroA sub-path changes, the related Binding SID 4001 is constant

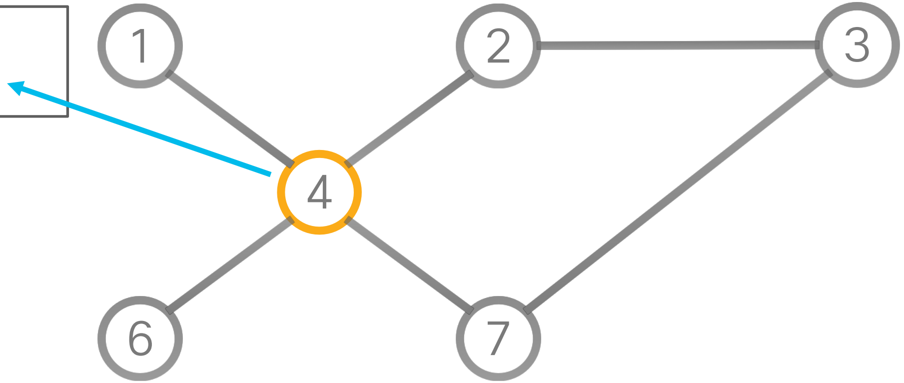
SR Data Plane Monitoring (SR-DPM)

SR-DPM: Data Plane Monitoring

- Check the true dataplane status of all FIB entries
  - For all linecards
  - For all ECMP paths
- Scalability
  - Each router tests its own dataplane
- Incremental deployment
  - Local router behavior
  - Only neighbor requirement: supports IGP Adj SID

Configuration

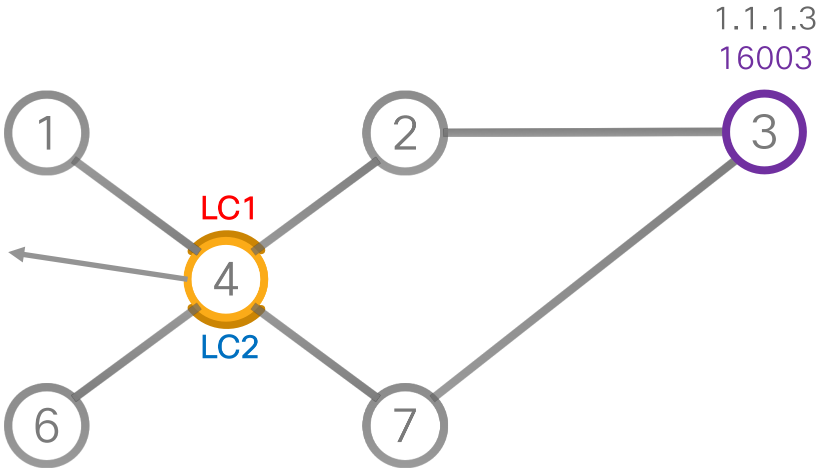
```
mpls oam
 dpm
```



Example: SR-DPM @ 4 for entry 16003

FIB (LC1):
In_lbl 16003
out 16003, to R2
out 16003, to R7

FIB (LC2):
In_lbl 16003
out 16003, to R2
out 16003, to R7



- 4 needs to test independently all its 4 FIB entries to 16003 (R3)
- Idea: 4 crafts a probe to deterministically test each entry

A 2-stage process that repeats every 30min

- Stage1 – Adjacency Validation
  - Validate the IGP Adj SID of each neighbor

- Stafe 2 – Dataplane FIB entry Validation
  - Leveraging the IGP Adj SID of each neighbor and the local knowledge of the ECMP hash, craft a test packet that will go to the neighbor and come back to visit the intended linecard for the intended prefix and for the intended ECMP hash
  - Put enough info in the test packet to remember which linecard, destination and ECMP path was tested
  - Set the TTL to expire at the downstream neighbor
  - Leverage the normal TTL-expiry behavior of the neighbor

Adj-SIDs validation

R4:

Adj to R1 ✔
Adj to R2 ✔
Adj to R6 ✔
Adj to R7 ✔

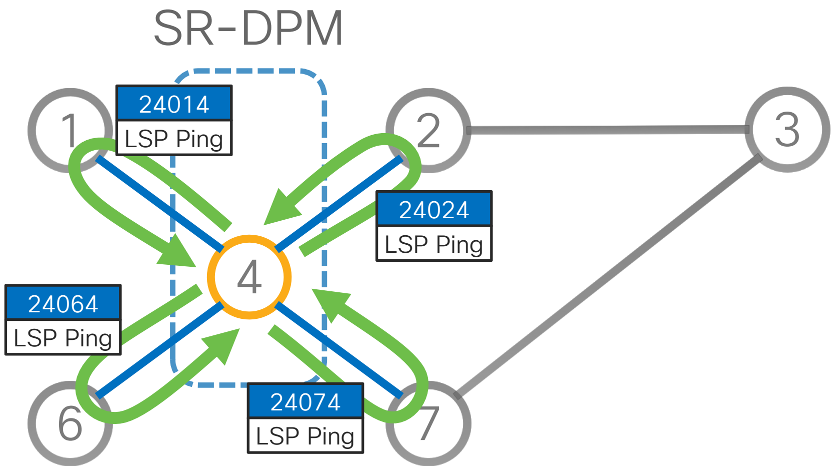


Illustration convention:
Adj-SID Rx-Ry: 240xy

✔ 4 tests its top linecard for ECMP path via 2 for FIB 16003

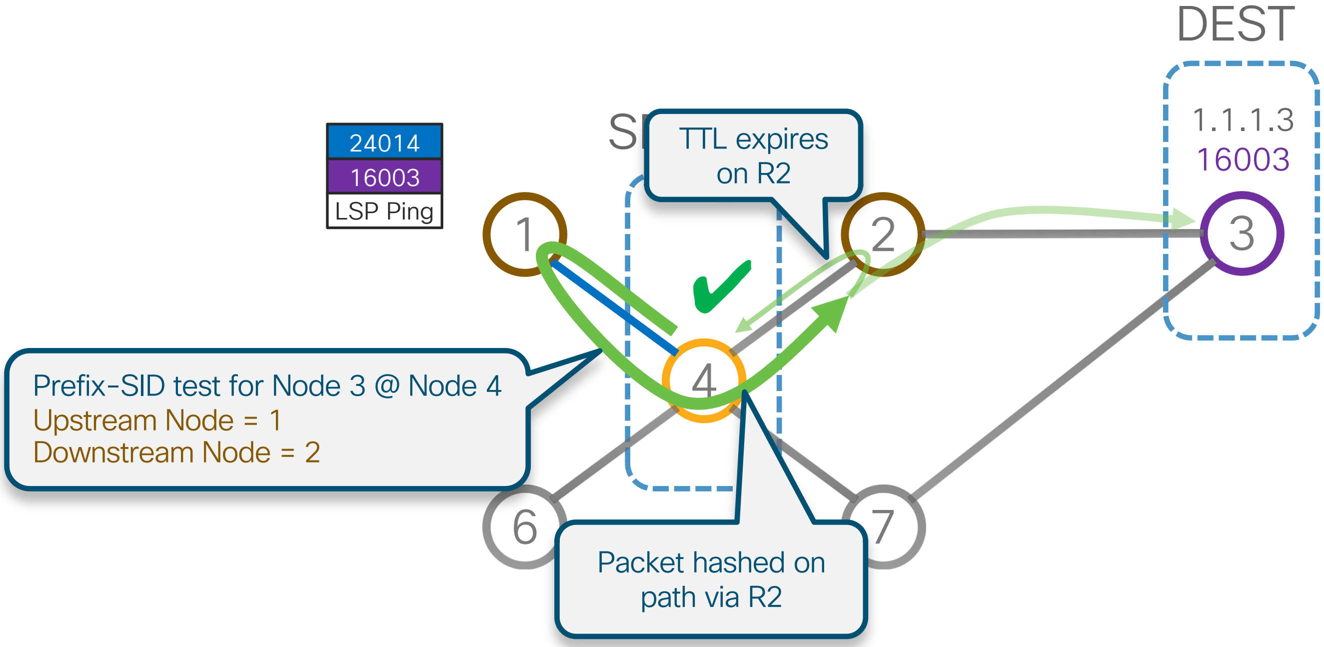


Illustration convention:
Adj-SID Rx-Ry: 240xy

✔ 4 tests its top linecard for ECMP path via 7 for FIB 16003

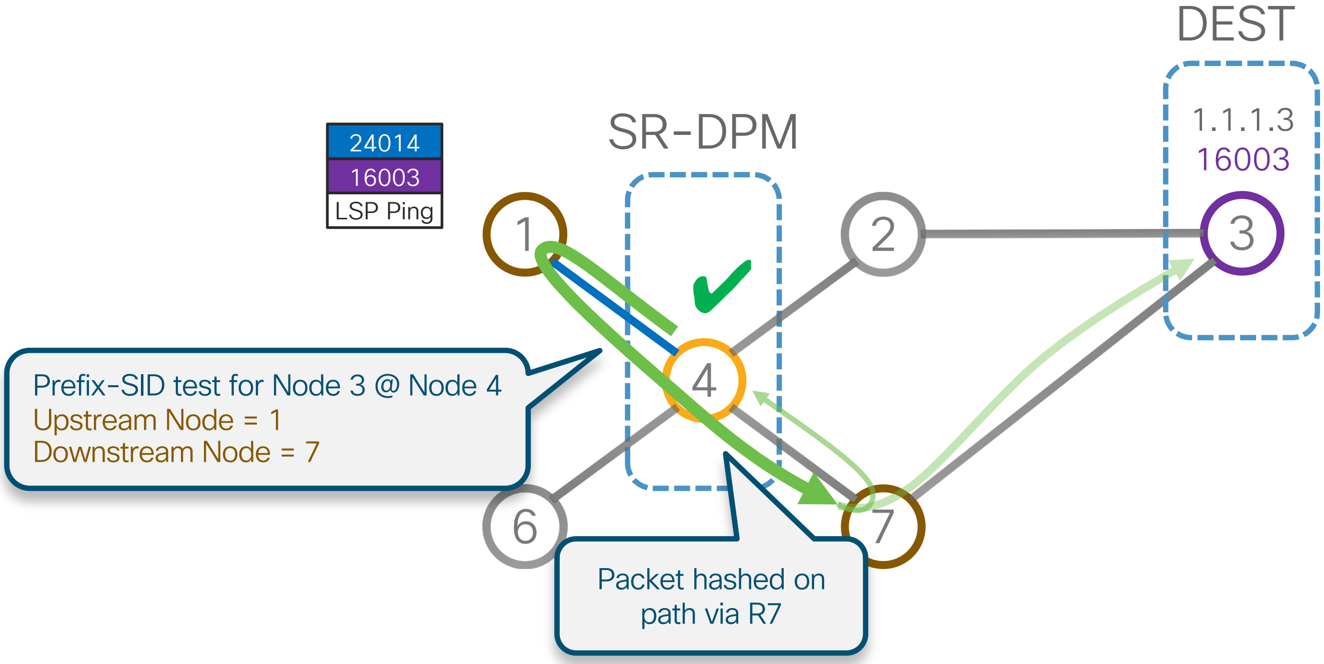


Illustration convention:
Adj-SID Rx-Ry: 240xy

✔ 4 tests its bottom linecard for ECMP path via 2 for FIB 16003

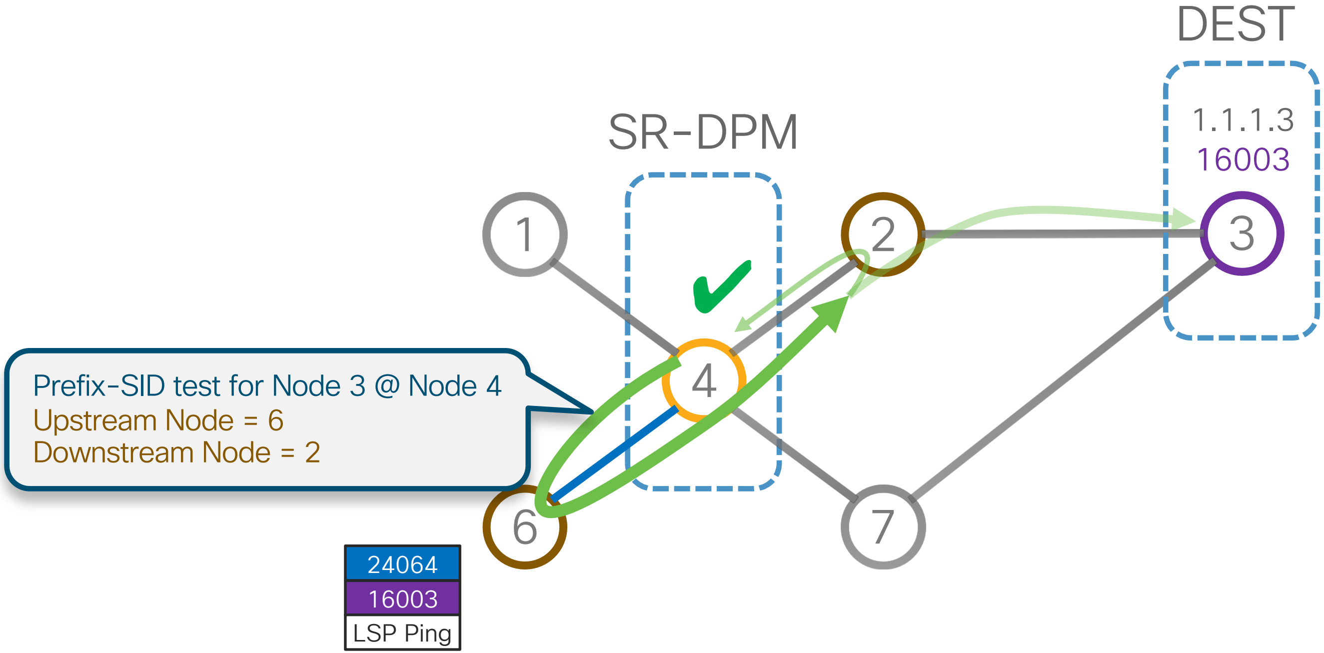
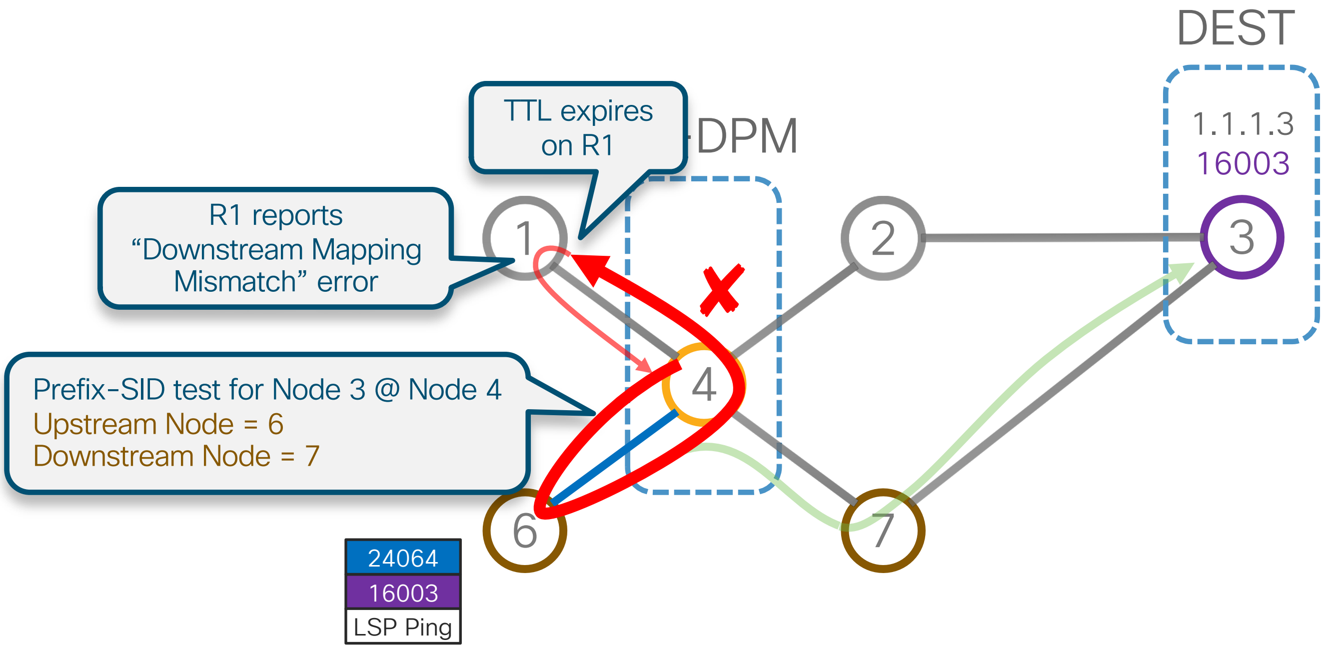


Illustration convention:
Adj-SID Rx-Ry: 240xy

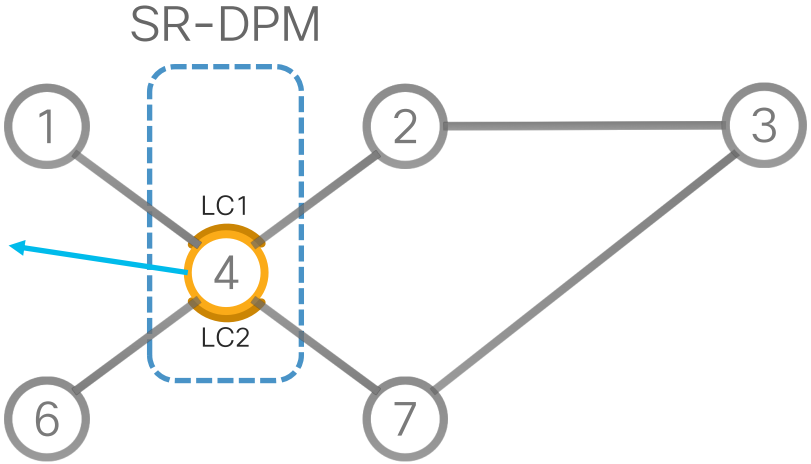
✗ 4 tests its bottom linecard for ECMP path via 7 for FIB 16003



- Dataplane FIB corruption: this entry wrongly points to intf to 1
- DPM logs detailed syslog error message for detected faults

SR-DPM @ 1 for 16003

FIB (LC1):	DPM:
In_lbl 16003	
out 16003, to R2	✔
out 16003, to R7	✔
FIB (LC2):	
In_lbl 16003	
out 16003, to R2	✔
out 16003, to R7	✘



- DPM keeps results of its verifications
- Without DPM this error could stay undetected for weeks

Conclusion

- A unique and innovative approach to tackle data plane consistency verification and traffic black hole detection challenges.
- Overcomes scale challenges by distributing the detection process, while still achieving validation of entire customer traffic path.
- Interoperable by design – no special processing needed at neighbors. No standardization required at IETF.
- Complements existing OAM solutions instead of replacing them.

SR-MPLS Conclusion

Industry at large backs up SR

Strong customer adoption
WEB, SP, DC, Metro, Enterprise

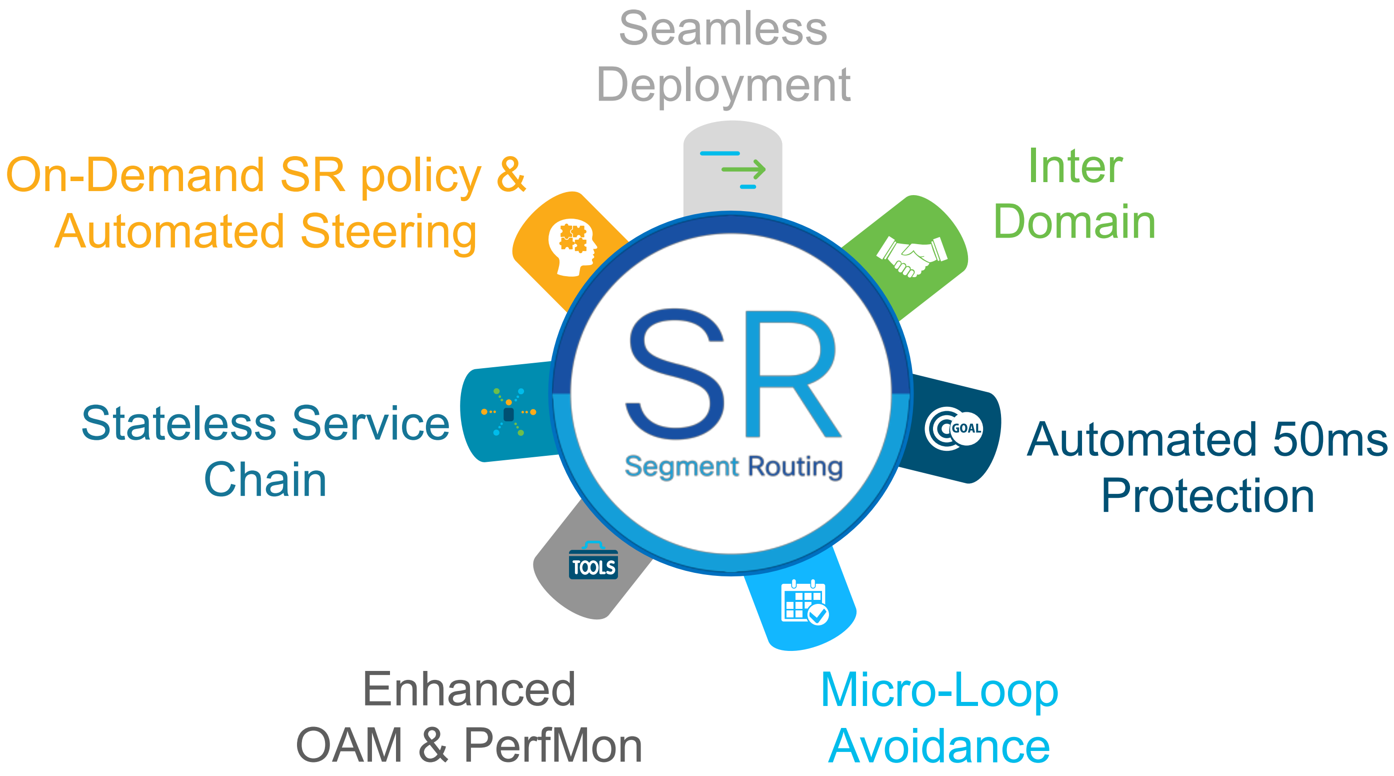
De-facto SDN Architecture

Standardization
IETF

Multi-vendor Consensus

Open Source
Linux, VPP

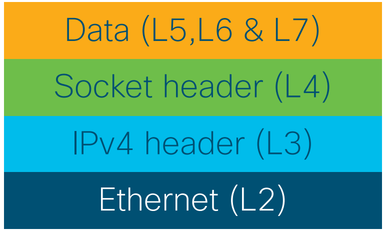
SR Unified Fabric Attributes



SRv6

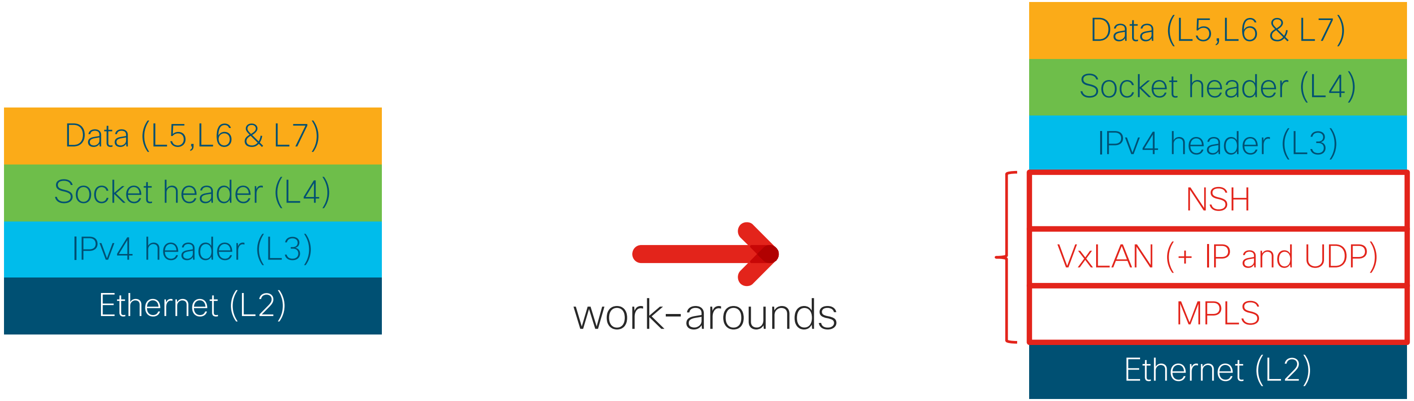
IPv4 limitations

- × Limited address space
- × No engineered Load Balancing
- × No VPN
- × No Traffic Engineering
- × No Service Chaining



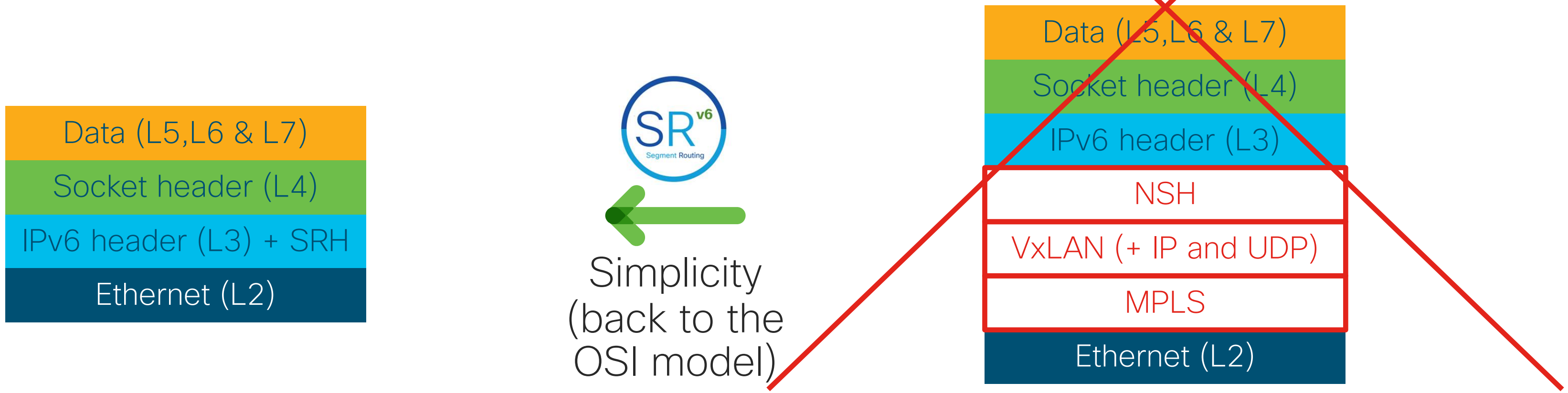
IPv4 limitations & work-arounds

- × Limited address space → NAT
- × No engineered Load Balancing → MPLS Entropy Label, VxLAN UDP
- × No VPN → MPLS VPN's, VxLAN
- × No Traffic Engineering → RSVP-TE, SR-TE MPLS
- × No Service Chaining → NSH

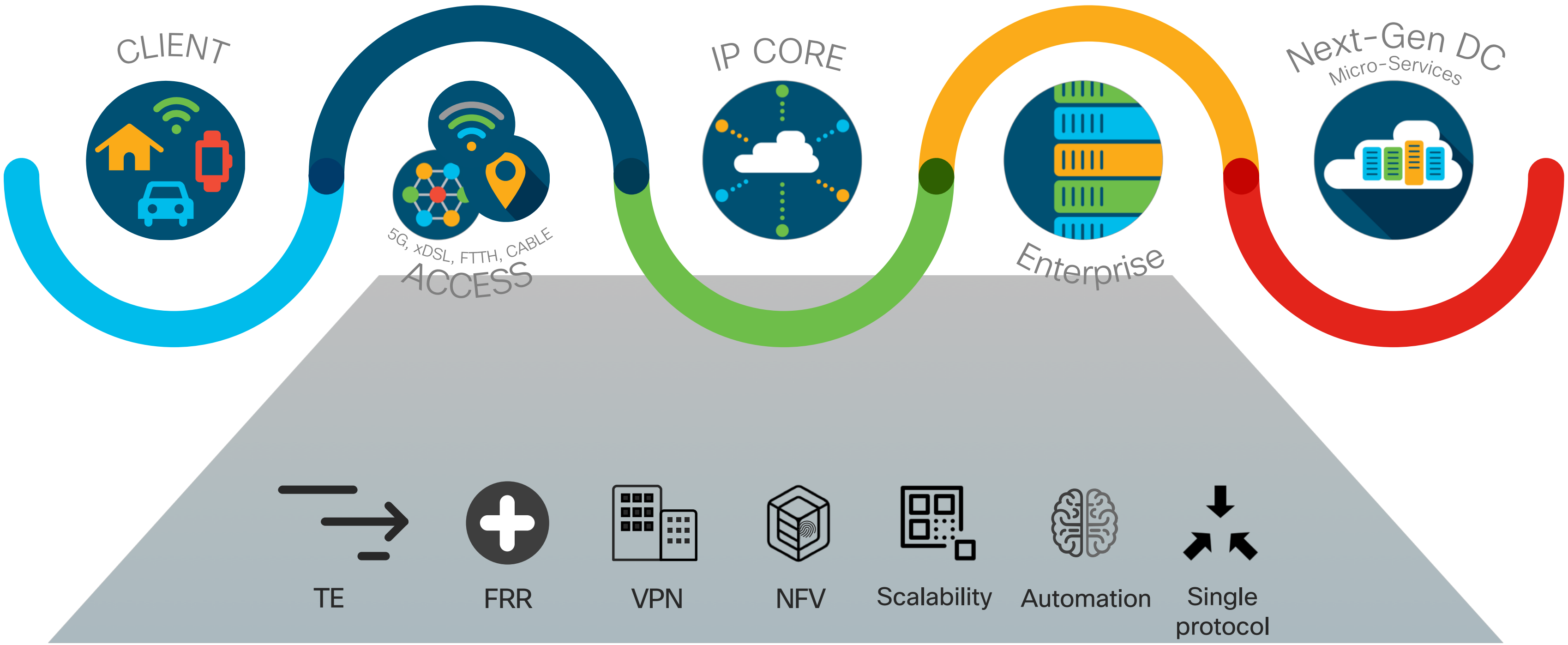


SRv6 Solution

- ✓ 128-bit address space → NAT
- ✓ IPv6 flow label → MPLS Entropy Label, VxLAN UDP
- ✓ SRv6 VPN → MPLS VPN's, VxLAN
- ✓ SRv6 Traffic Engineering → RSVP-TE, SR-TE MPLS
- ✓ SRv6 Service Chaining → NSH



SRv6 unleashes IPv6 potential

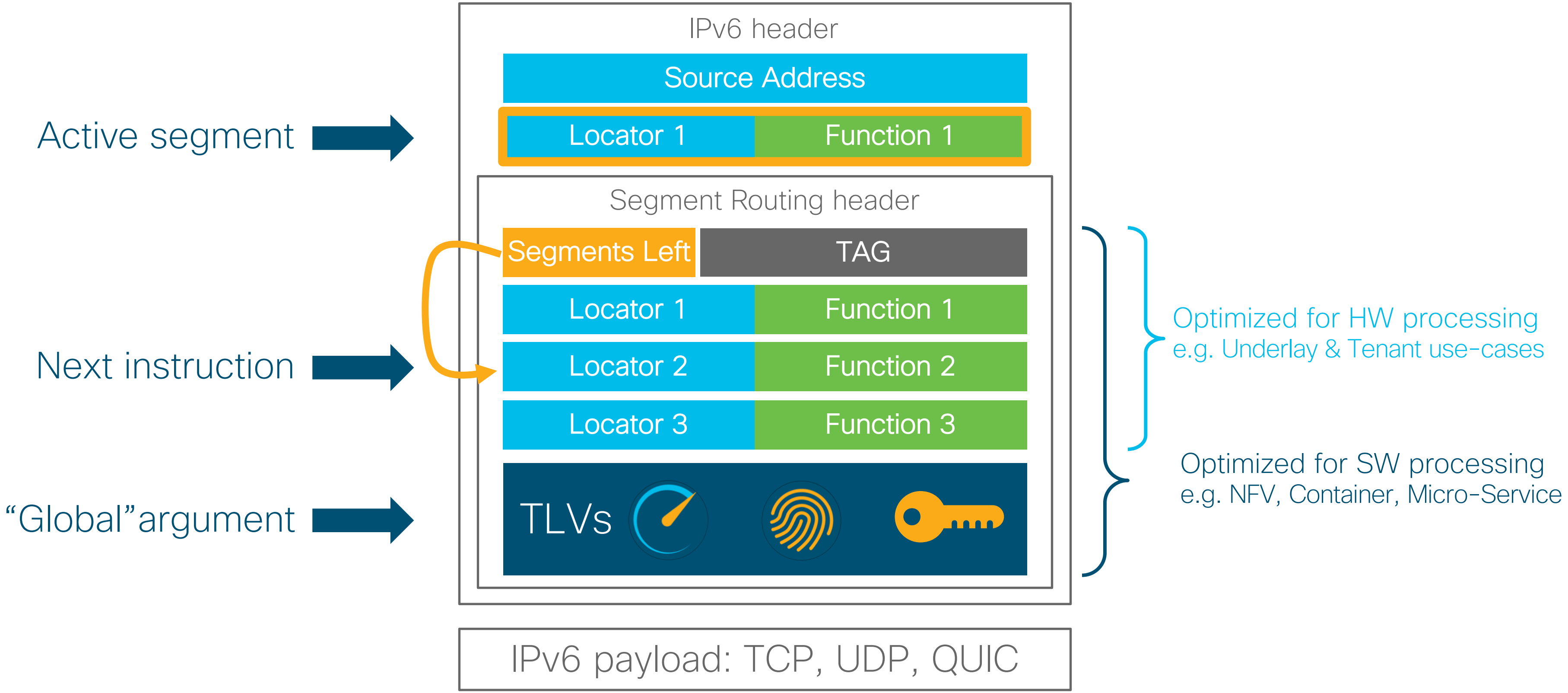


SRv6 fundamentals

Network instruction

- 128-bit SRv6 SID
  - Locator: routed to the node performing the function
  - Function: any possible function
    either local to NPU or app in VM/Container
  - Flexible bit-length selection

Network Program in the Packet Header



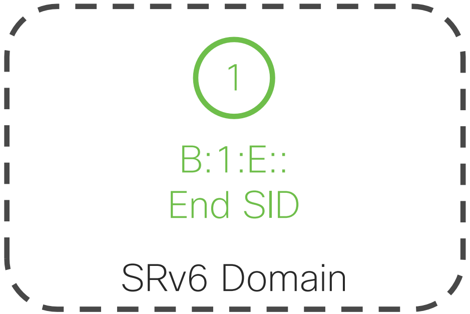
End and End.X SID behaviors

Illustration convention:

- IPv6 address of node k is A:<k>::
- SRv6 SID of node k is B:<k>:<function>::

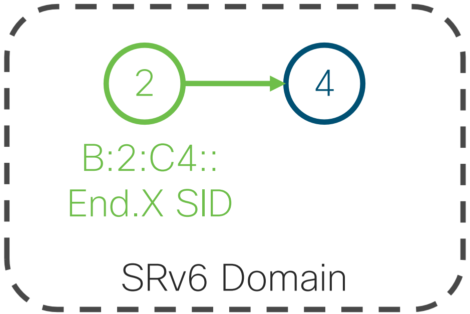
- End – Default endpoint behavior
  - shortest-path to the SID's endpoint
  - endpoint updates DA with next SID
  - endpoint forwards according to updated DA

Illustration convention B:<k>:E::



- End.X – Endpoint with cross-connect
  - shortest-path to SID's endpoint
  - endpoint updates DA with next SID
  - endpoint forwards to interface associated with SID

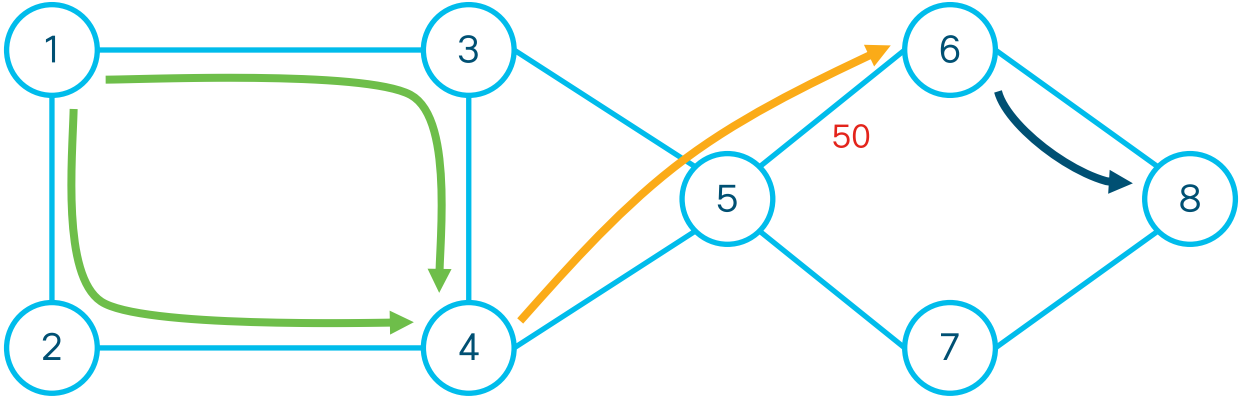
Illustration convention B:<k>:C<j>::, where j identifies the remote node



Endpoint behaviors illustration

Illustration convention:
- IPv6 address of node k is A:<k>::
- SRv6 SID of node k is B:<k>:<function>::

SR: ⟨ B:4:E::, B:5:C6::, A:8:: ⟩



Default metric 10

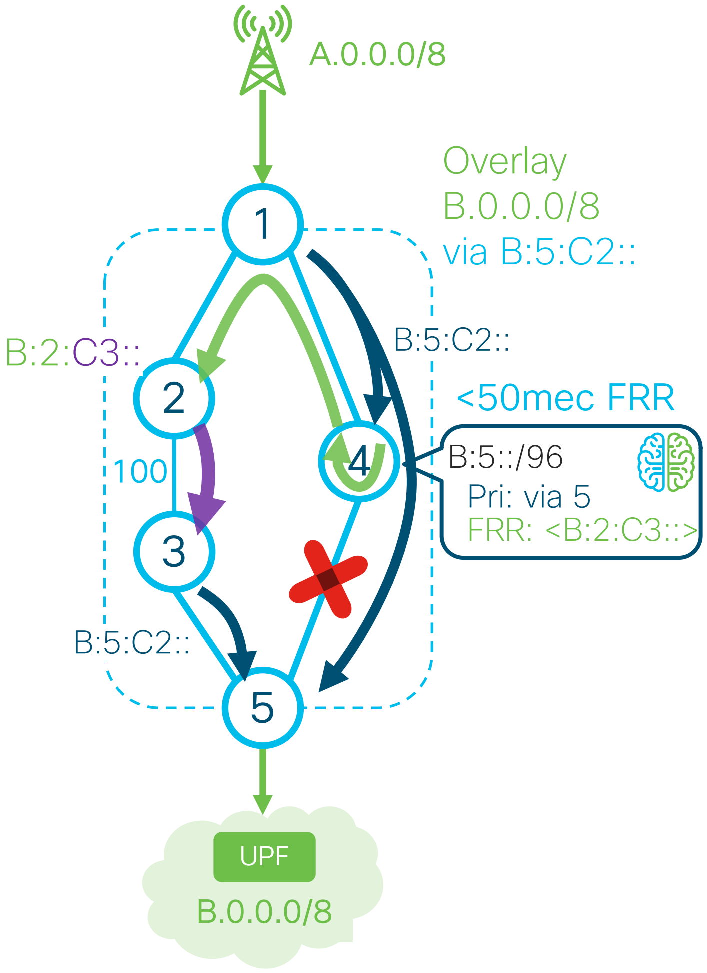
- B:4:E:: shortest path to node 4
- B:5:C6:: shortest path to node 5, then cross-connect towards 6
- A:8:: regular IPv6 address of node 8

Lead Operators and Academia

SRv6 TILFA

High Availability - TILFA

- 50msec Protection
- Link, node or SRLG failure
- Automated hence Simple
- Per-Destination Optimum backup path
- Incremental deployment
- No new protocol
  - lightweight extension to ISIS/OSPF



Cisco Implementation

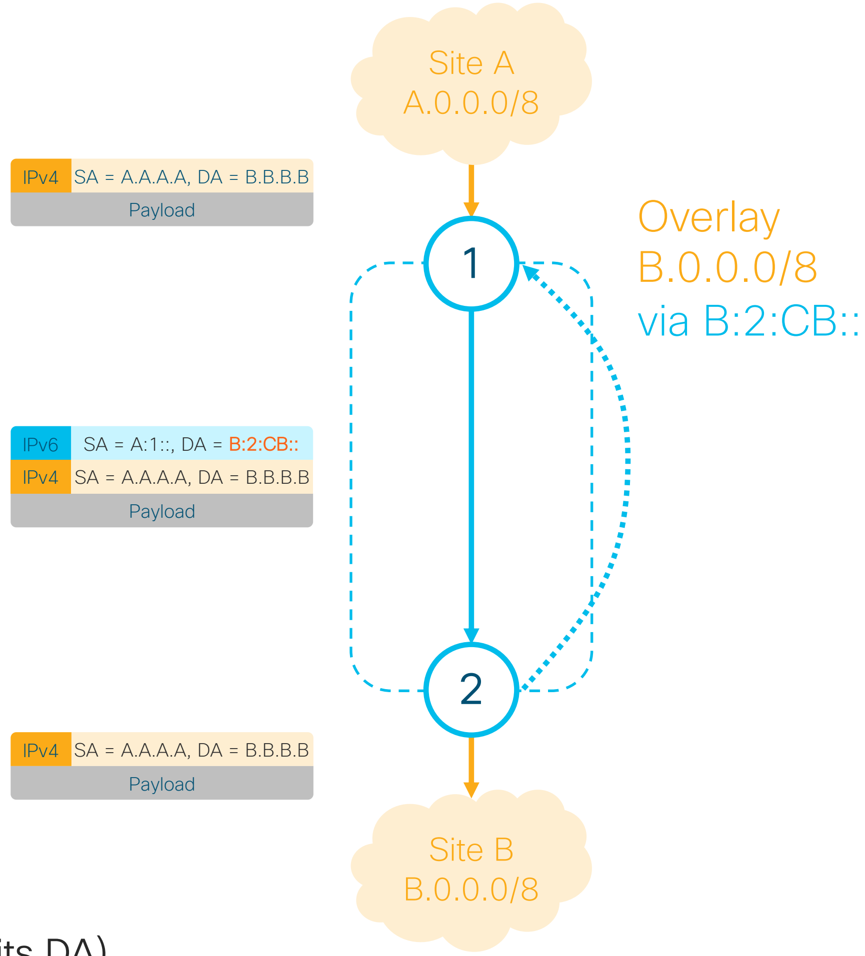
- FCS since 2014 for SR-MPLS
  - Numerous deployments
- FCS since Dec 2018 for SRv6
  - Deployment in early CY19

```
router isis <inst>
   interface <>
     address-family ipv6 unicast
       fast-reroute per-prefix ti-lfa
     !
   !
!
```

SRv6-VPN

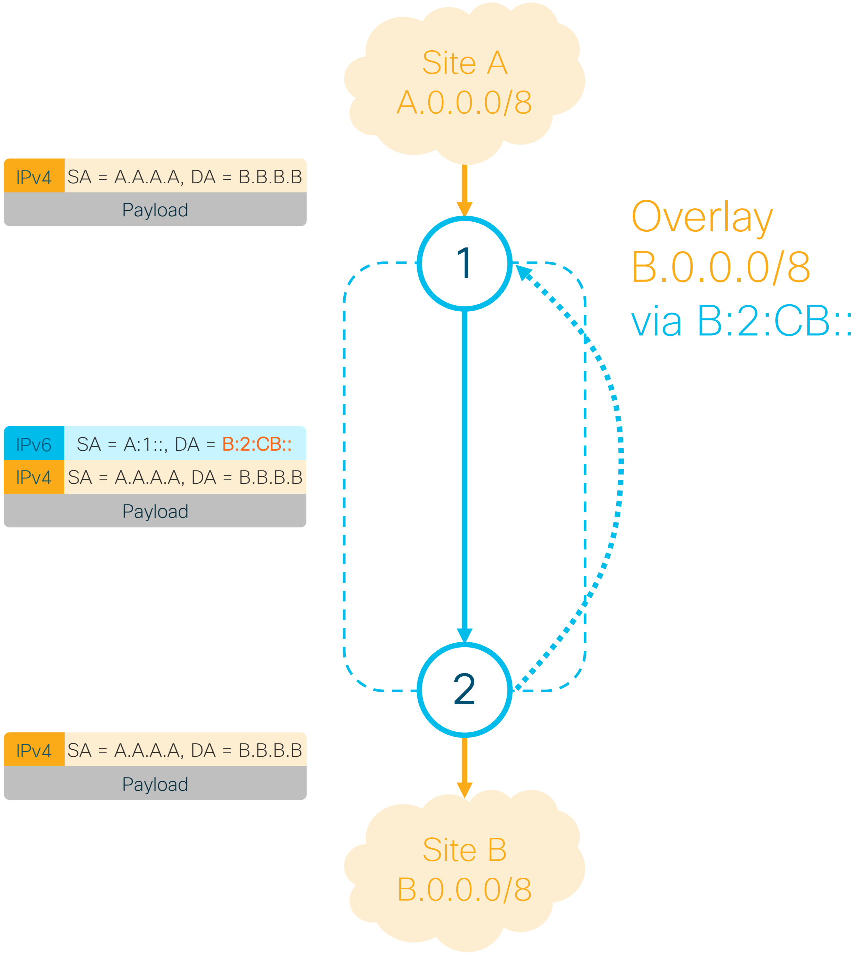
Overlay

- One single SID is needed
  - B:2:CB
  - "go to 2, decaps and lookup in VRF B"
- No new procol (just BGP)
  - No new SAFI
  - Light ext. to BGP Prefix-SID attribute
- Automated
  - No tunnel to configure
- Efficient
  - SRv6 for everything
  - No other protocol, just IPv6 with SRv6
    - > In fact, SRH not even needed (one single SID fits DA)



Overlay configuration

```
router bgp <inst>
  address-family vpnv4 unicast
    segment-routing srv6
      locator <name>
    !
  !
  neighbor <ipv6-addr>
    address-family vpnv4 unicast
    !
  !
  vrf <>
    address-family ipv4 unicast
      segment-routing srv6
        alloc mode {per-vrf | per-ce}
      !
  !
!
```

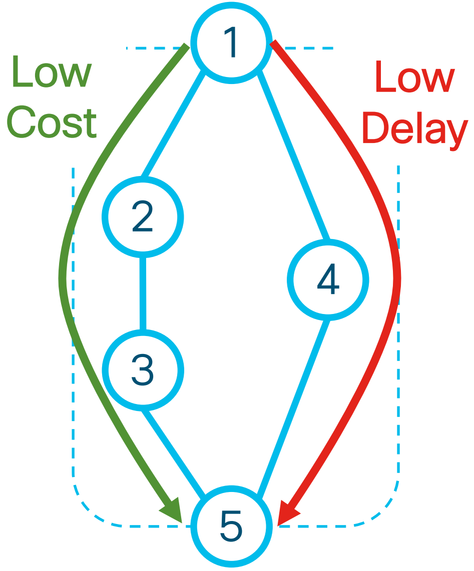


Implementation

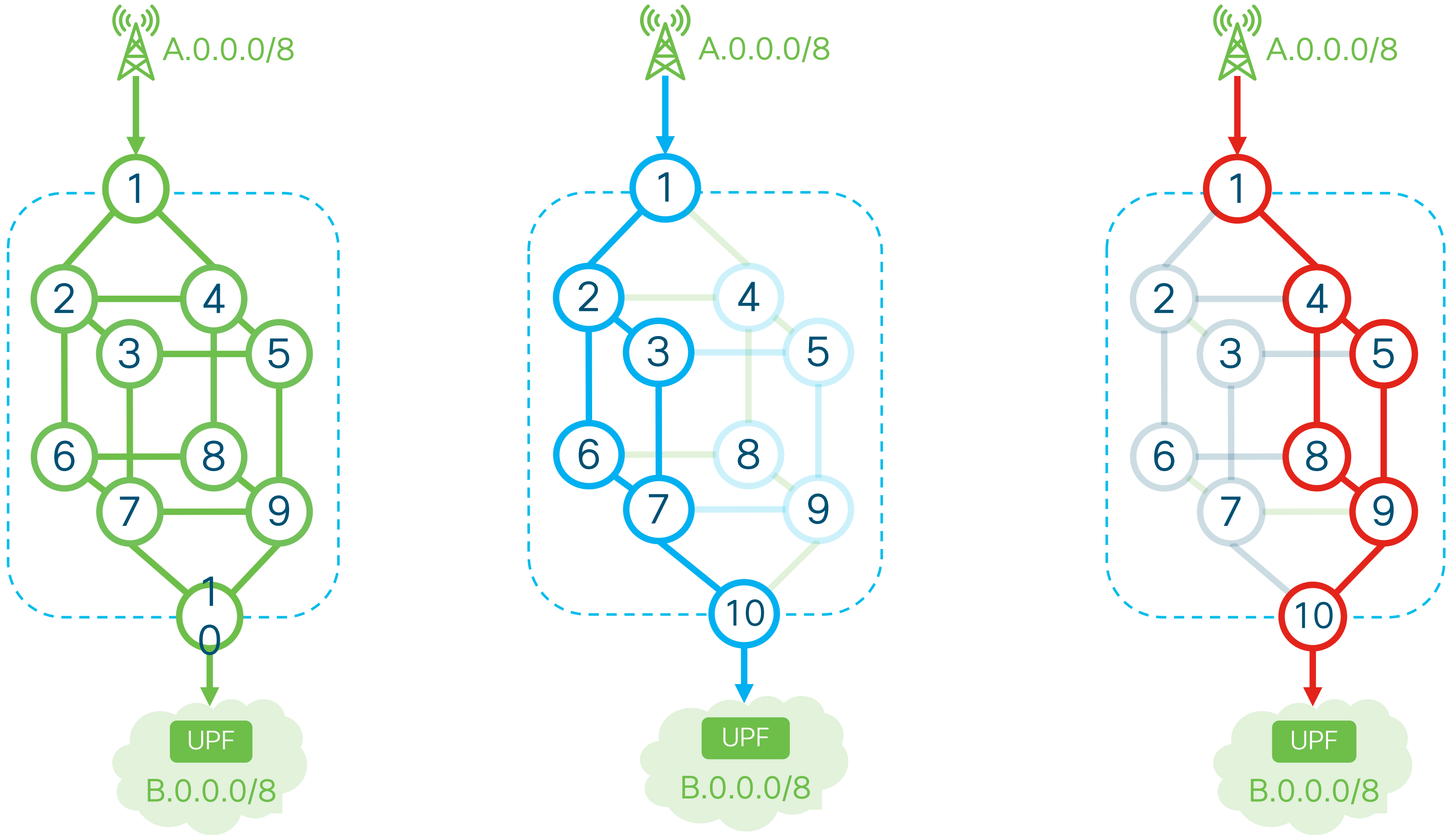
- Cisco HW
  - FCS since Dec 2018
  - Deployment early CY19
- Open-source
  - Dataplane for VPP and Linux
  - BGP/RIB client for VPP

SRv6 Flex-Algo

SRv6 Flex-Algo: Low-Cost vs Low-Latency



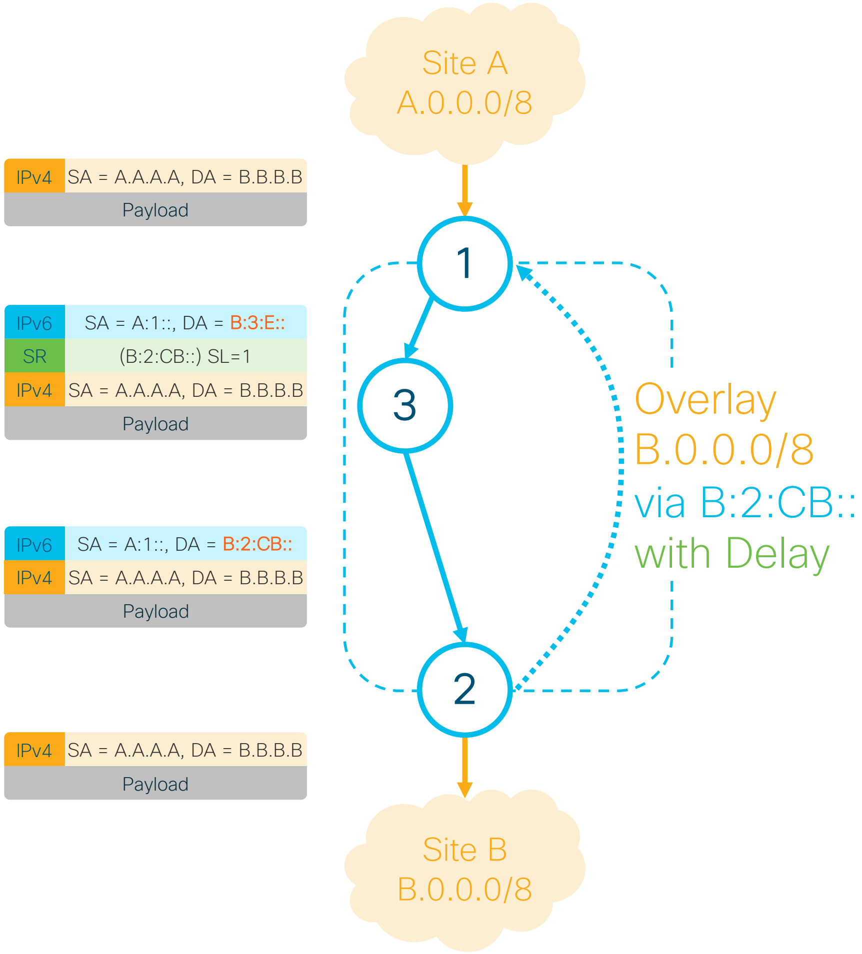
SRv6 Flex-Algo: Any Plane vs Left Plane vs Right Plane



Integrated
VPN + TE + NFV

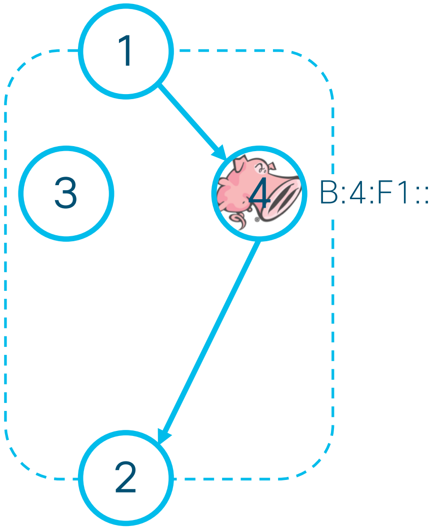
VPN + TE

- Network programming concept
  - <B:3:E::, B:2:CB>
- Go to 3 for low-delay TE
- Go to 2, decaps and get in VRF B
- SRH holds the network program



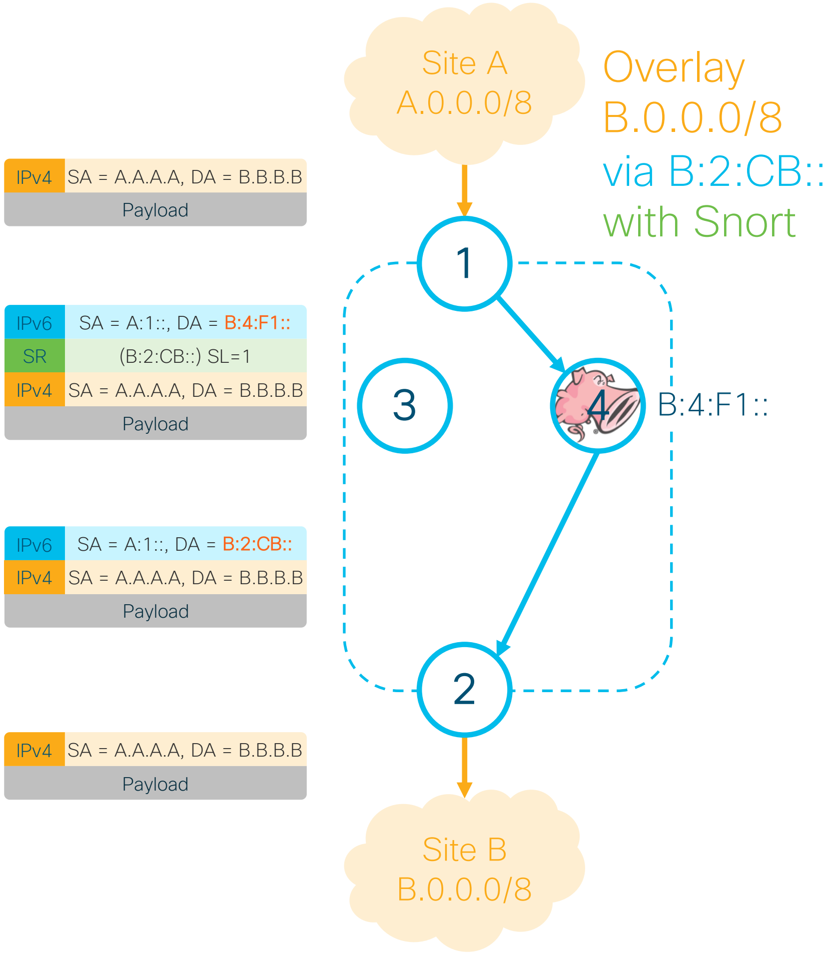
Network Function (NF)

- B:4:F1
  - Go to 4 and get through Network Function F1
    - E.g. F1 is Snort container
- Stateless NFV without any new protocol
  - (NSH is one more stateful protocol – bad)
- NF can leverage the SRH
  - Implement branching operation
  - Read / write metadata
- Open-source SR-aware NFs
  - Snort, iptables, nftables
  - Leverage native SRv6 support in Linux kernel



VPN + NF

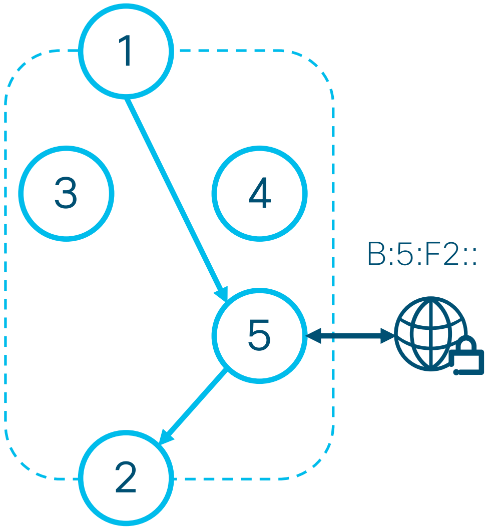
- Network programming concept
  - <B:4:F1::, B:2:CB>
- Go to 4 and get Snort function
- Then go to 2, decaps and get in VRF B
- SRH holds the network program



SR proxy functions for Legacy NFs

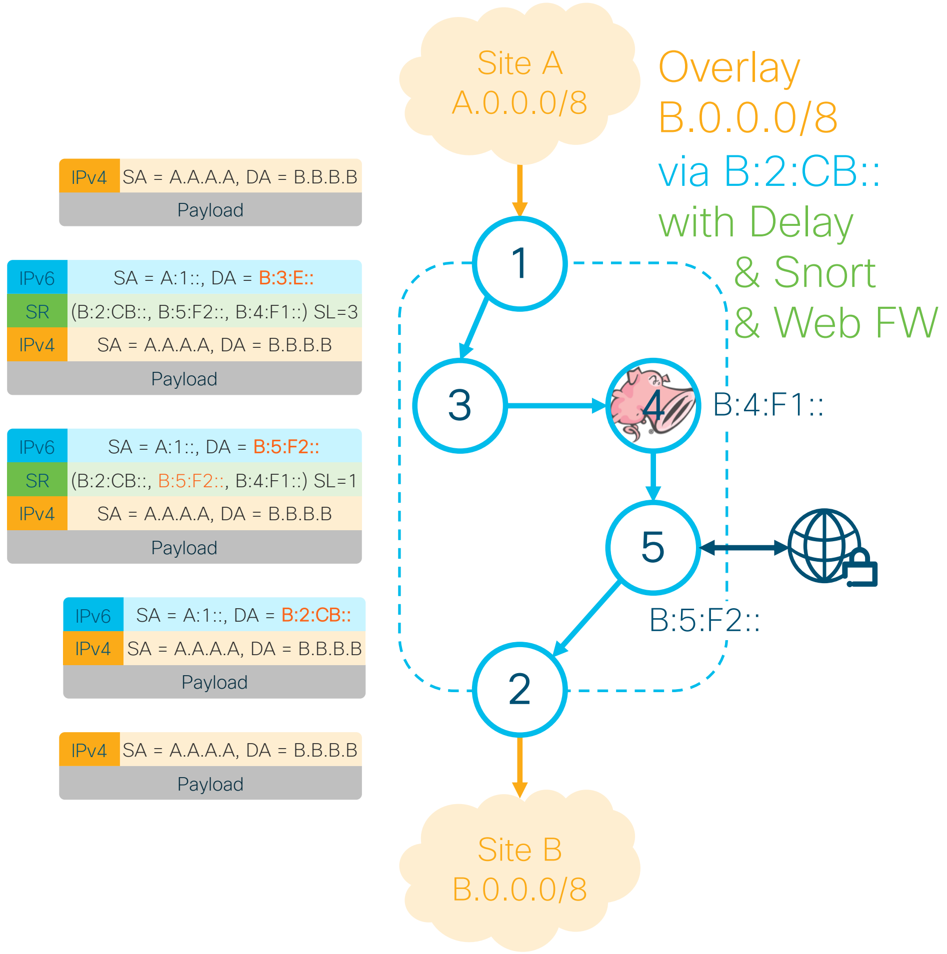
- B:5:F2
  - Go to 5 and get an SR Proxy Function
- Proxy processes the SRH
- Legacy NF processes the inner packet

- Works with any physical or virtual NF
  - Support IPv4, IPv6 and L2 traffic
- Open-source proxy implementations
  - FD.io VPP and Linux



VPN + TE + NF

- Network programming concept
  - <B:3:E::, B:4:F1::,B:5:F2:: B:2:CB>
- Go to 3 for low-delay TE
- Go to 4 and get Snort function
- Go to 5 and get legacy Web app
- Go to 2, decaps and get in VRF B
- SRH holds the network program



No New Protocols, No State

- No new Dataplane protocol
  - <> NSH which is a new stateful protocol just for NFV
- No new Control Plane protocol
  - Leverage the SRTE Control Plane without any change ☺
- Stateless (as all the SR solution)

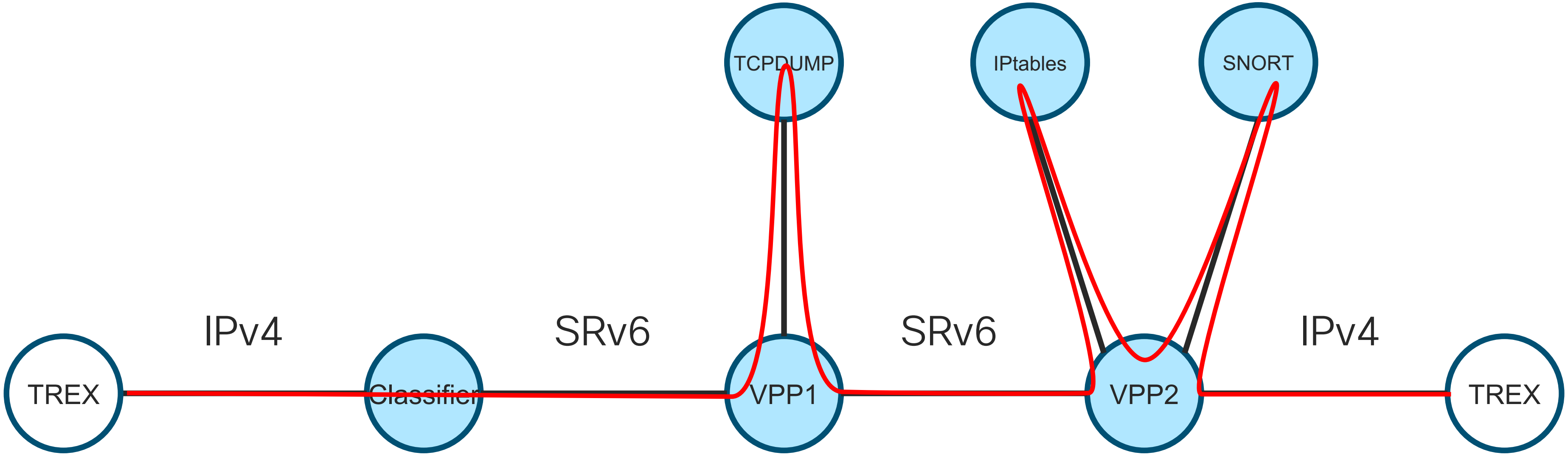
- This is why we say “Service Programming”
  - Much better solution than “Service Chaining” with PBR/NSH

Cisco Implementation

- IOS-XR and IOS-XE implementations available
- Open-source VPP
- Open-source Linux
- Open-source SR-aware applications

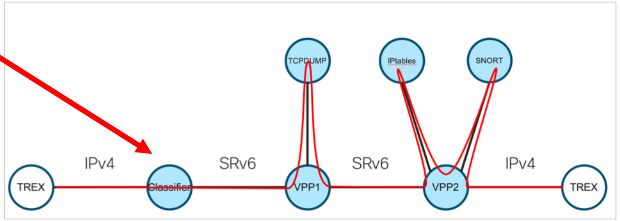
SRv6 Service Programming with Metadata

Network Topology



Classifier

ENEA
Qosmos Division



- DPI, classifies traffic basedon traffic patterns
- Encapsulates traffic into IPv6
- Destination address is SRv6 function:

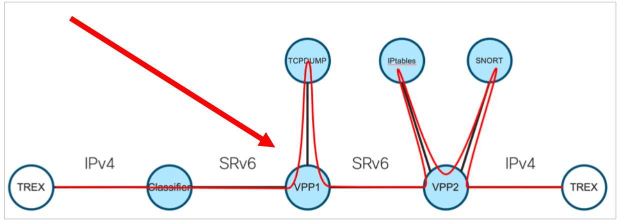
Locator Function

B:1:C::XXXX

Argument

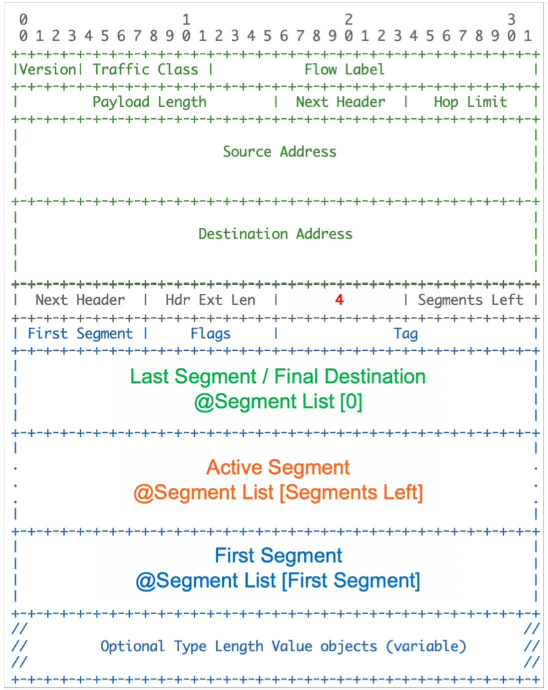
- Argument identifies real protocol
- 0004 HTTP
- 00C6 SSH

VPP1

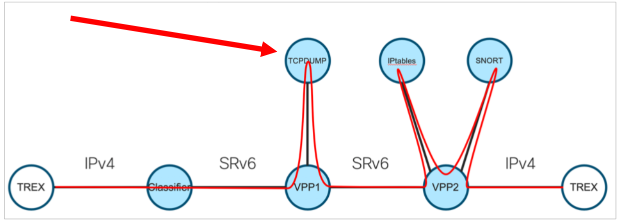


- Binding SID for Service chain: B:1:C::XXXX
- Service Chain:
  - B:1:C100:: END.X towards TCPDUMP
  - B:200:A:: SRv6 aware IPtables can read TLVs
  - B:300:A:: SRv6 aware SNORT
  - B:2:D4: END.DX4 egress

  - Argument is translated to OPAQUE TLVs



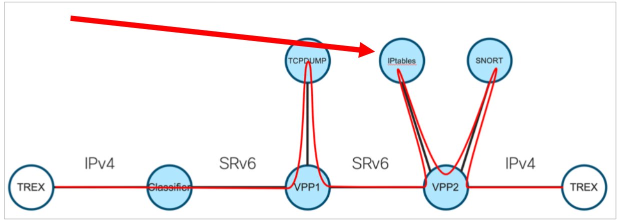
TCPDUMP



- Capturing Packet with SRH and TLV
- Not able to decode TLV

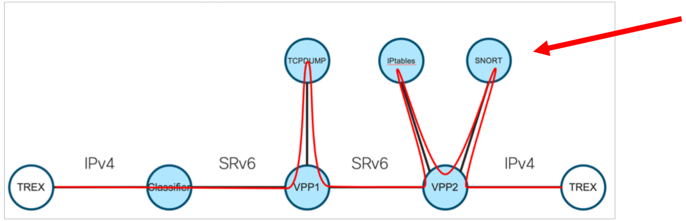
```
Address[0]: b:2:d4::
Address[1]: b:300:a:: [next segment]
Address[2]: b:200:a::
Address[3]: b:1:c100::
.......................

0060  02 00 00 0a 00 00 00 00 00 00 00 00 00 00 00 0b   ................
0070  00 01 c1 00 00 00 00 00 00 00 00 00 00 00 06 0e   ................
0080  c6 00 00 00 00 00 00 00 00 00 00 00 00 00 45 00   ..............E.
```

Firewall



- SRv6 v 6 aware IPTABLES
- Can do action on encapsulated packets
- Can drop packets based on TLV
- `ip6tables -I FORWARD -m srh --srh-opaque-tlv 0xc6000000000000000000000000000 -j DROP`

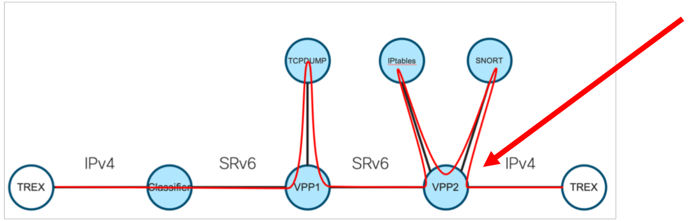
SNORT



- SRv6 aware
- Can inspect packet encapsulated in SRv6
- Alerting if port 22 is seen

```
01/29-02:51:29.840865  [**] [1:5000:0] ALERT! 10.0.2.99 -> 10.0.1.99  [**]
[Priority: 0] {TCP} 10.0.2.99:1025 -> 10.0.1.99:22
```

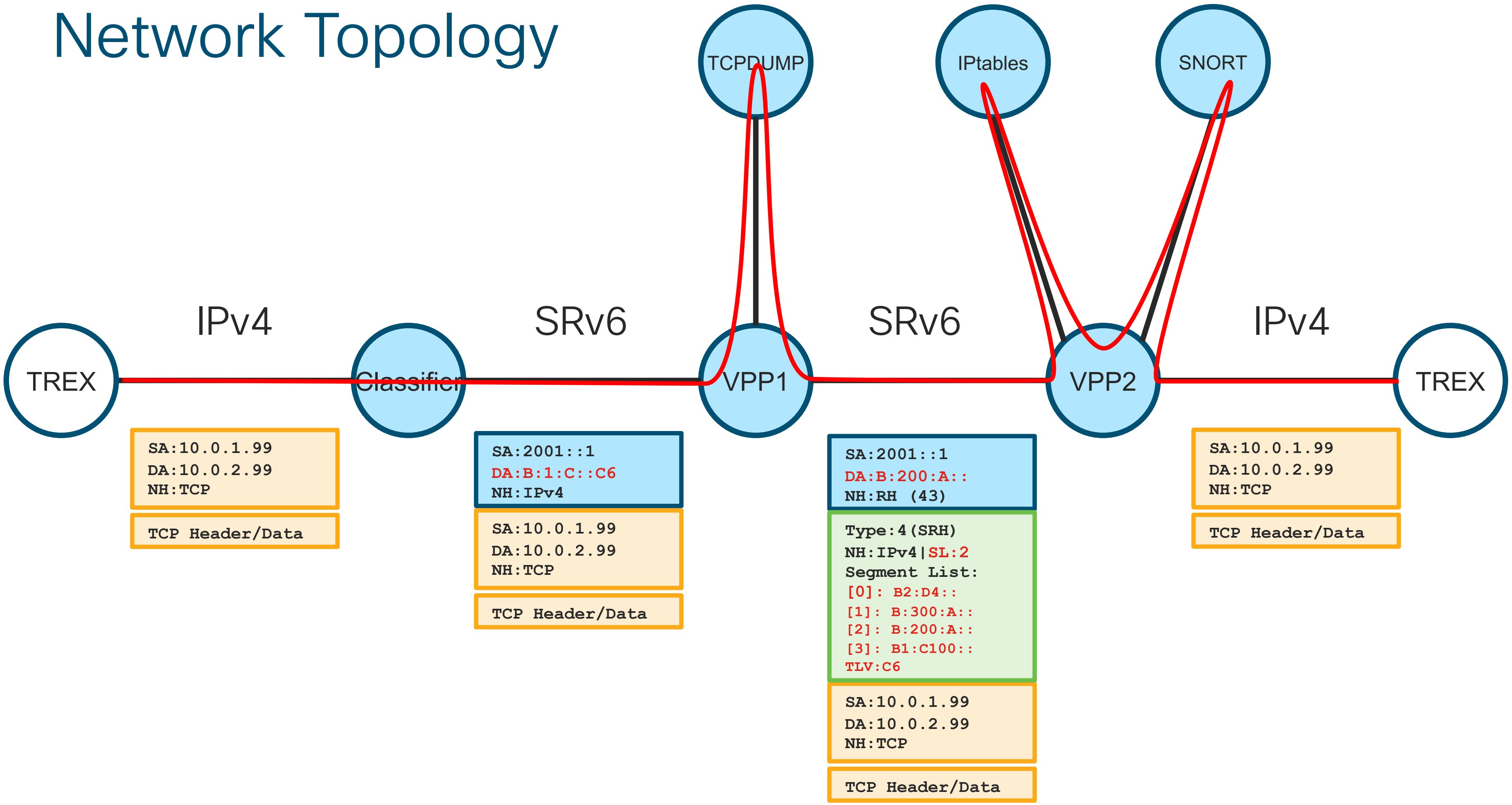
VPP2



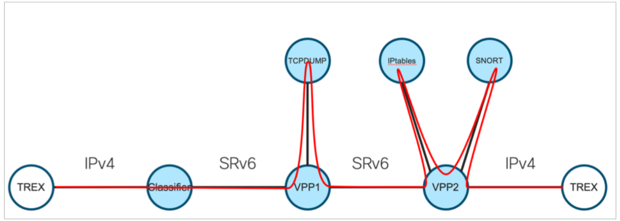
- Decapsulates packets END.DX4
- IPv4 packets are send towards traffic generator

```
01/29-02:51:29.840865  [**] [1:5000:0] ALERT! 10.0.2.99 -> 10.0.1.99  [**]
[Priority: 0] {TCP} 10.0.2.99:1025 -> 10.0.1.99:22
```

Network Topology



Summary



- We demonstrated SRv6 service programming
- QoSMoS classifier encodes metadata in the SRH
- SNORT and IPTables FW deliver their NF while leveraging the application ID encoded in the metadata
- No new dataplane or control plane protocol
- No state
- The power of simplification with SRv6

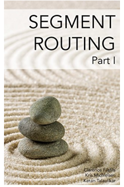
Stay Up-To-Date

http://www.segment-routing.net/

https://www.linkedin.com/groups/8266623

https://twitter.com/SegmentRouting

https://www.facebook.com/SegmentRouting/



Segment Routing, Part I - Textbook

CISCO

IMAGINE

INTUITIVE

5G and Network Slicing

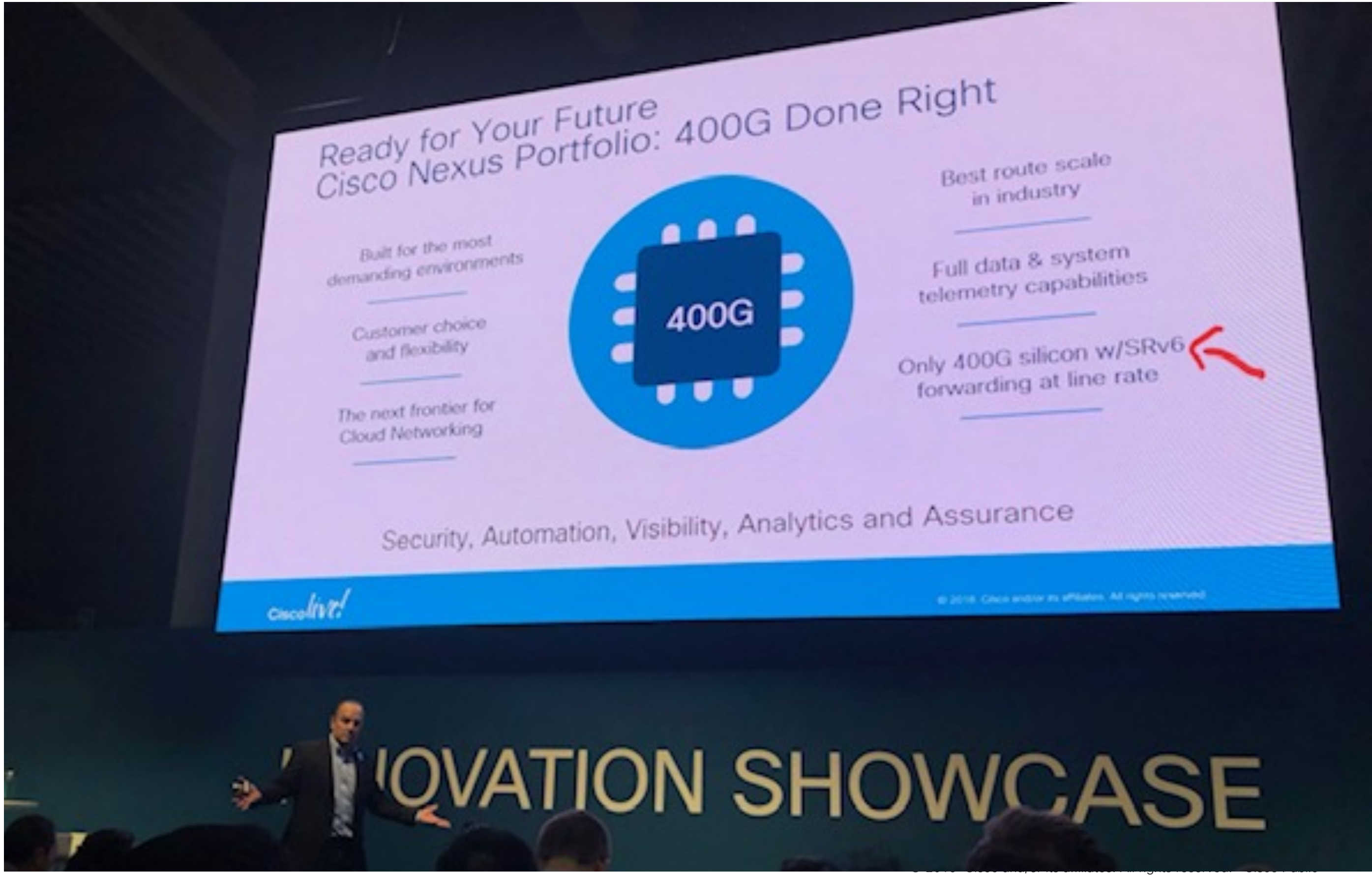
SR: a simple and complete solution for 5G

- ✓ Scale
- ✓ High Availability
- ✓ VPN
- ✓ Ultra-Low Latency
- ✓ OAM and Performance Monitoring
- ✓ DC / VNF

SRv6 Data Center

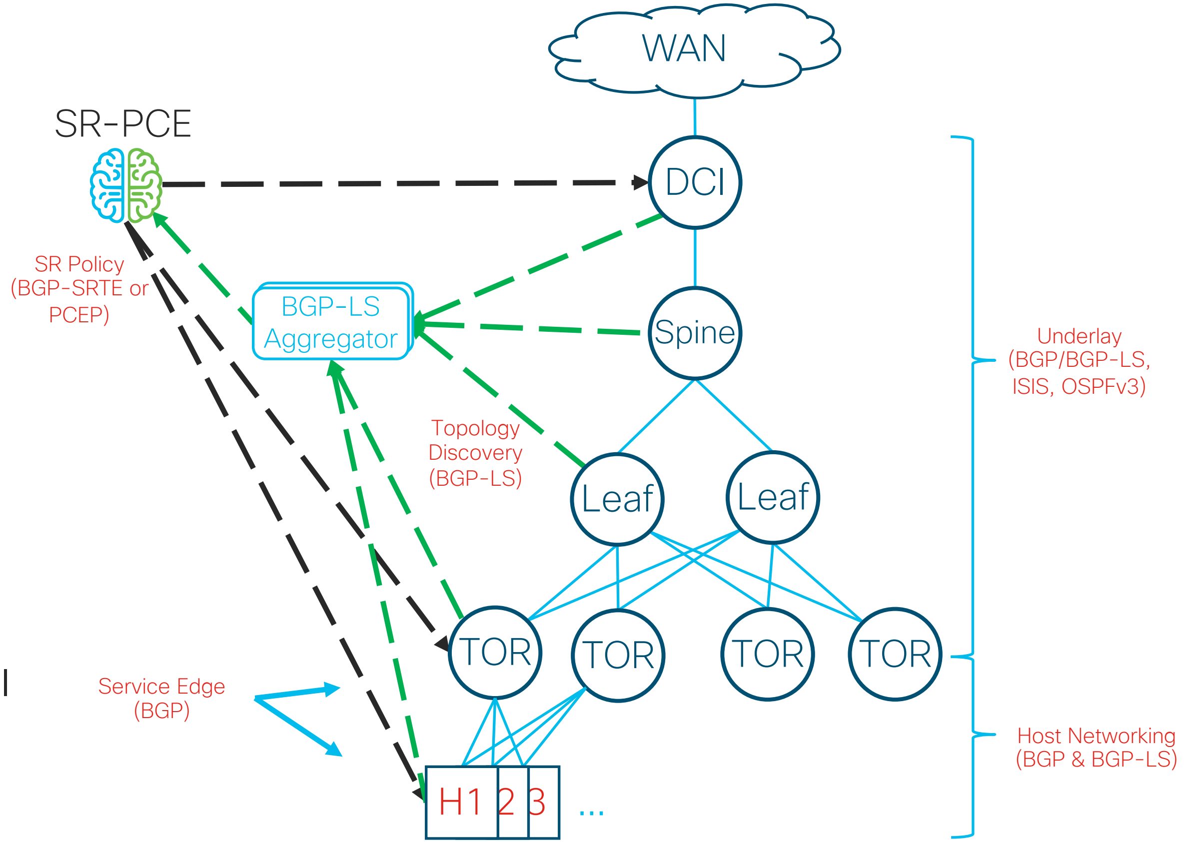
DC/Nexus: Leadership SRv6 @ 400G

- Amazing set of SRv6 network instructions @ 400G !



SRv6 Data Center

- Underlay
  - One IPv6 domain per DC
- Overlay
  - End-to-end services with BGP and SRv6
- SRTE
  - There are intra-DC use-cases
  - The source of the end-to-end policy is often in the DC
- Host Networking
  - SRv6 capabilities on Bare Metal or Virtualized Compute



Where are we?

Cisco live!

SRv6 endpoint behaviors in Linux and VPP

Name	Description	Linux kernel	VPP
End	Default endpoint	4.10 (Feb 2017)	17.04 (Apr 2017)
End.X	Layer-3 cross-connect	4.10 (Feb 2017)	17.04 (Apr 2017)
End.T	Specific IPv6 table lookup	4.14 (Nov 2017)	17.10 (Oct 2017)
End.DX2	Decapsulation and Layer-2 cross-connect	4.14 (Nov 2017)	17.04 (Apr 2017)
End.DX4	Decapsulation and IPv4 cross-connect	4.14 (Nov 2017)	17.04 (Apr 2017)
End.DX6	Decapsulation and IPv6 cross-connect	4.14 (Nov 2017)	17.04 (Apr 2017)
End.DT4	Decapsulation and specific IPv4 table lookup	-	17.04 (Apr 2017)
End.DT6	Decapsulation and specific IPv6 table lookup	4.14 (Nov 2017)	17.04 (Apr 2017)
End.B6.Insert	Endpoint bound to an SRv6 policy	4.14 (Nov 2017)	17.04 (Apr 2017)
End.B6.Encaps	Endpoint bound to an SRv6 Policy with encapsulation	4.14 (Nov 2017)	17.04 (Apr 2017)
End.AS	Static proxy to SR-unaware application	-	18.04 (Apr 2018)
End.AD	Dynamic proxy to SR-unaware application	srext module	18.04 (Apr 2018)
End.AM	Masquerading proxy to SR-unaware application	srext module	18.04 (Apr 2018)
End.BPF	Endpoint bound to an arbitrary eBPF program	4.18 (Aug 2018)	-
T.Insert	Transit with insertion of an SRv6 Policy	4.10 (Feb 2017)	17.04 (Apr 2017)
T.Encaps	Transit with encapsulation in an SRv6 Policy	4.10 (Feb 2017)	17.04 (Apr 2017)
T.Encaps.L2	Transit with encapsulation of L2 frames	4.14 (Nov 2017)	17.04 (Apr 2017)

Custom SRv6 behaviors with eBPF (End.BPF)

- Associates local SRv6 SID with user-defined eBPF program
  - Leverage Extended Berkeley Packet Filter (eBPF) functionality of the Linux kernel
  - User-defined C function inserted into the networking pipeline at run-time
  - No kernel compilation required
  - Guaranteed stability
- Provides helper functions to
  - Apply basic SRv6 behaviors (End, End.X,...)
  - Steer traffic into an SR policy
  - Add, modify or delete TLVs
- Available in Linux kernel 4.18 (August 2018)

SRv6-aware applications

- Wireshark (June 2016)
- pyroute2 (October 2017)
- tcpdump (December 2017)
- iptables (January 2018)
- SERA firewall (January 2018)
- nftables (March 2018)
- Snort (March 2018)

SRv6 Conclusion

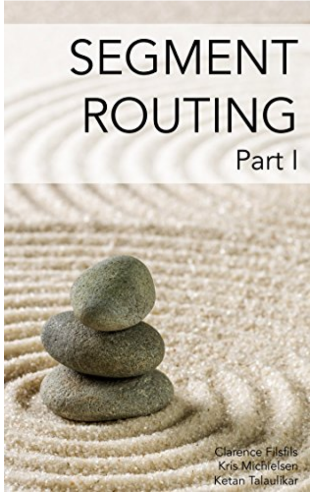
Simplicity always prevails

SRv6 Status

- Strong Lead Operator team
- Comprehensive IETF definition
- Open-Source: linux and VPP
- Industry consensus and Inter-Operability: barefoot, unistarcom…
- VPN and TILFA FCS ☺
- First deployments with Cisco ☺

Stay up-to-date

amzn.com/B01I58LSUO



segment-routing.net

linkedin.com/groups/8266623

twitter.com/SegmentRouting

facebook.com/SegmentRouting/

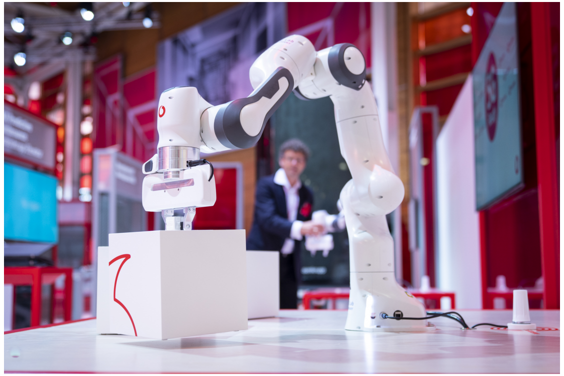
Demo 1

Dynamic latency optimization in Service Provider network

Problem statement | 5G is coming...

Tele-operated driving

Hologram call



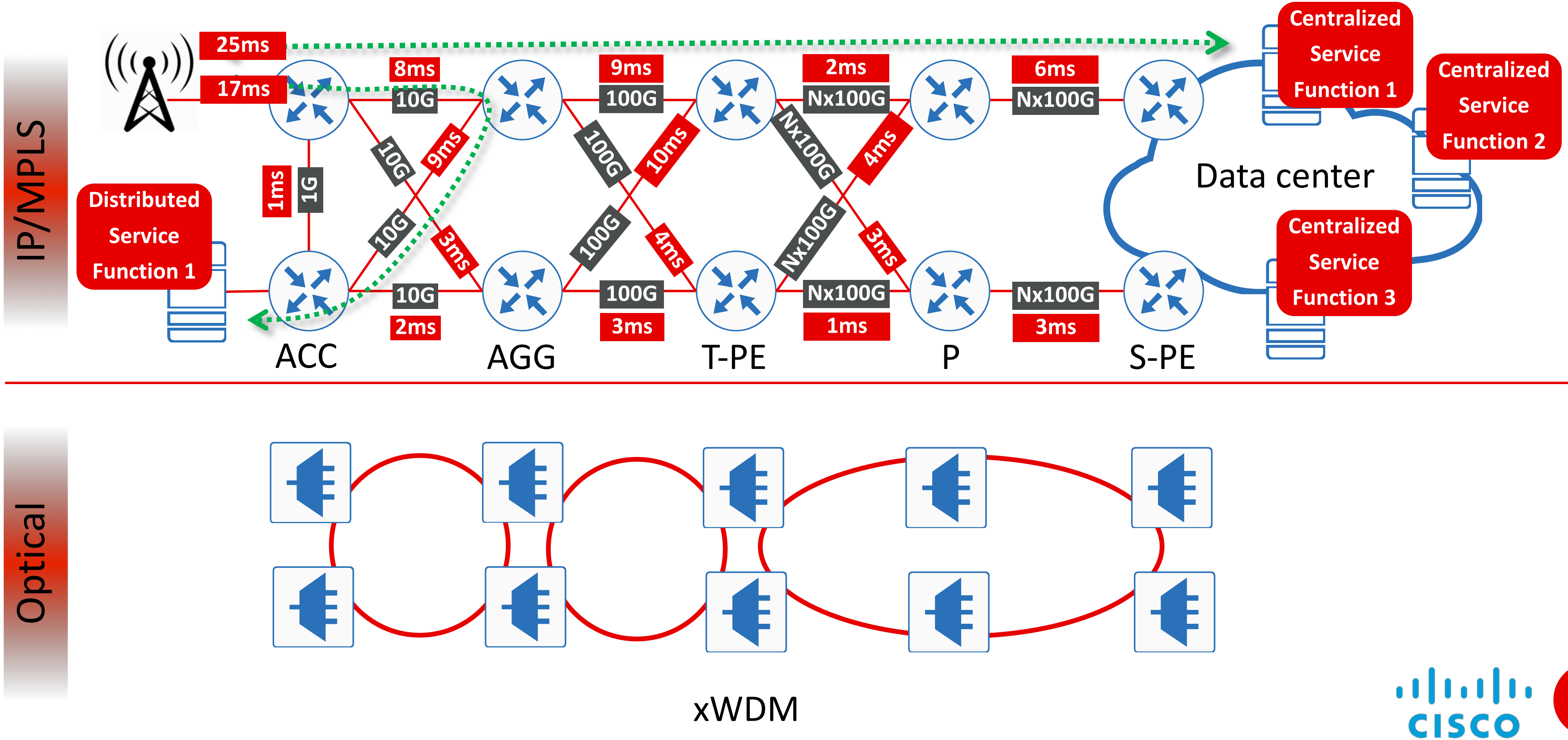
Real-time robotics

Low end-to-end latency for application

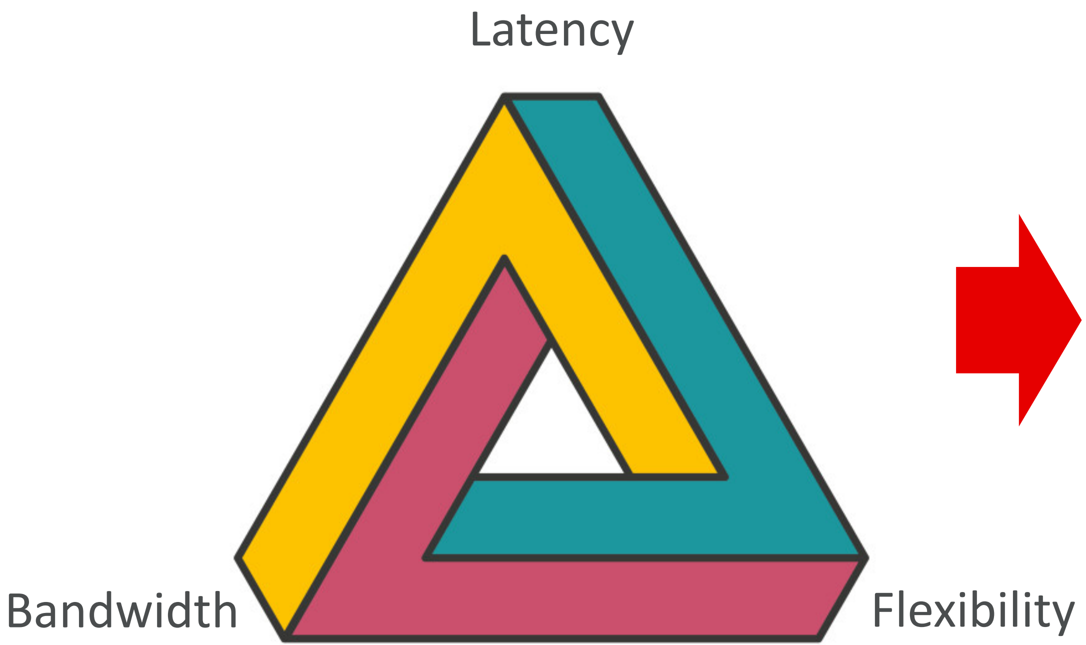
Problem statement | Is traditional routing good enough?

Standard routing is based on OSPF/ISIS metrics

Doesn't matching 5G latency requirements

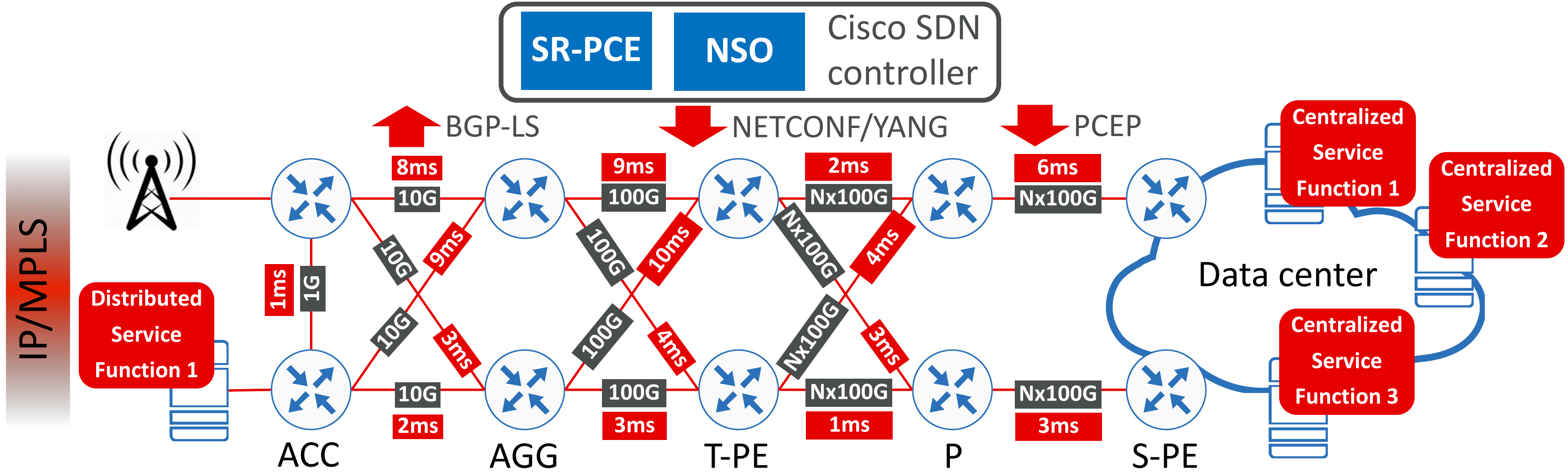


Problem statement | 5G requires something different



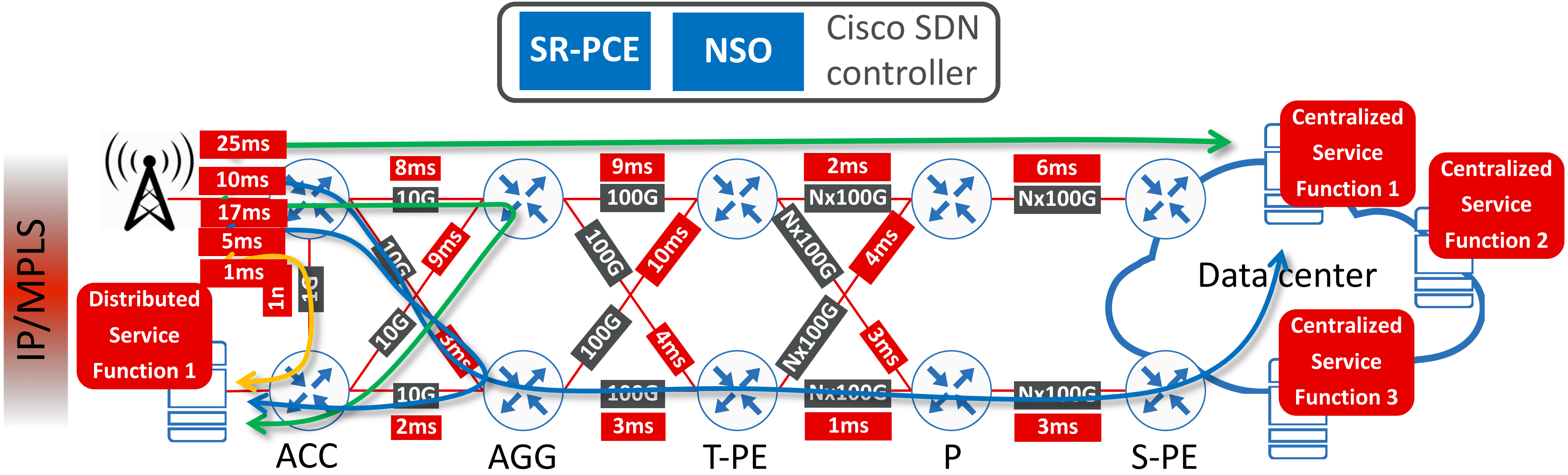
How to get all 3 together?

Solution | Latency optimization with delay boundary



- Collect information about network topology and export it to SR-PCE using BGP-LS
- Measure per-link latency and export it to SR-PCE using BGP-LS as well
- Configure SR-TE policy with delay boundary at ACC/S-PE from NSO using NETCONF/YANG
- Push respective SR-TE LSP to ACC/S-PE from SR-PCE using PCEP

Solution | Optimize latency only where required



- Latency to central service function:
  - w/o optimization: 25 ms
  - boundary 10 ms: 10 ms
- Latency to distributed service function:
  - w/o optimization: 17 ms
  - boundary 10 ms: 5 ms
  - boundary 3 ms: 1 ms

Benefits

- This solution is on active trial within our network
- Latency optimization where necessary, BW optimization everywhere
- Closed-loop assurance for automated service restoration to guarantee end to end latency KPI

Key take away

- Our networks could be smarter than we can imagine
- SDN helps to prepare network for 5G
- Delay optimization with delay boundary is key transport network feature for slicing

Thank you!

IMAGINE

INTUITIVE